INF-3981
Master's Thesis in
Computer Science

Inferring Image Semantics from Collection Information

by

Idar Thorvaldsen

06-02-2009

Faculty of Science
Department of Computer Science
University of Tromsø

Contents

Chapter 1

Introduction

1.1 Objective

The goal of this project is to infer image semantics for individual images based on information about the collections to which the images belong. Information from these different collections will be combined to create a better description of individual images than is available from the collections they belong to seen separately. Various techniques will be used to combine the information of the collections when searching, leading to an improved description of image semantics. This combined semantic information will in turn be used in order to try to to improve image search performance. Hopefully, one can in gain an improved ability to search for images by better describing them through the process described above.

1.2 Motivation

There is a continuing desire and need for improving the processes of describing and searching for digital images. While good progress has been made adapting traditional information retrieval techniques to perform these tasks, processing images still presents a number of challenges not encountered when working with just text. Therefore, many different approaches have been and are being explored in order to improve the handling of image data in an information retrieval context. No single one of the existing solutions have proven to be sufficient, and it is unlikely that any one solution discovered in the foreseeable future will be the only one used.

A combination of approaches is therefore likely to yield better results than focusing on and using just one. The best system for image search is likely to be the one which can integrate several different techniques, both traditional information retrieval approaches as well as processes specifically developed for image search. Working towards a solution for just one of the challenges within image search is therefore of interest, even if in many cases this new solution would not perform better than others. As long as it has the potential to improve performance in some area, and can be combined with other techniques, it can be a possible contribution to the field.

Image search is becoming increasingly important, as the pace at which digital images are generated and used continues to increase. Several factors have contributed to this rapid growth: The Internet and the World Wide Web allowing for the easy transfer and publishing of digital images, improved bandwidth and image compression making near-instantaneous image transmission possible for most conventional users, and digital formats now being the norm for almost all new cameras of various types, to name a few. As the amount of digital information in general continues to increase, the amount of digital images will grow as well. However, if one does not have ways to search for these images, they are effectively almost impossible to find for anyone who does not already know of their existence.

1.3 Approach

The basic idea underlying this system is to not search for images directly, but instead first search for the collections they are part of. When a set of relevant collections have been found, one can then apply the collection-level semantics to the images belonging to these collections. An important concept in this project is that one image can be a part of several collections. Such images can be identified when collection information is added to the system, allowing one to link the same image to several collections before searches are run. Semantic information from several different collections can then be applied to the same image, potentially giving a better idea of what the image actually depicts.

This work fills a specific niche which has not been much explored until now. It allows for the indexing and searching of collections of images where images are not individually described. Such collections are typically largely unsuitable data material for traditional text-based image search systems. Images are likely to be part of collections of the type described above more often than other types of non-textual digital media. Also, since images are easy to copy and use, it is not uncommon for the same image to be used in more than one setting. This makes having multiple sources for the same image a possibility.

The techniques used in this project are by themselves quite rudimentary compared to the state of the art in information retrieval. It is primarily the way in which they are used which is of interest. The exact search and ranking algorithms employed are of secondary importance, and could easily be replaced with more advanced versions without really impacting the core aspects of this work. Also, this system will not handle the actual work of generating context descriptions for collections based on their context data; that is a task far beyond the scope of the current project. There has already been done much research into extracting the semantics of image contexts and similar data, and such systems are widely implemented in search engines and other information retrieval systems. A way of automatically processing context information will be necessary if the system is to be used on any real scale, of course, but for now, test data will be generated using a combination of simple algorithms and manual data entry based on existing usage contexts.

1.4 Related work

1.4.1 Image search

There is, of course, a large number of systems for performing image search already available, both well-established ones as well as more experimental systems. Commercial systems operate both on the web and are used in local settings, such as on home computers. Meanwhile, there are a vast number research projects both ongoing and completed that attempt to tackle the challenge of searching among images in various ways. The majority of research now seems to be in trying to examine the contents of images directly (using what is known as Content-Based Image Recognition, or CBIR), rather than looking at the textual information surrounding them.

However, CBIR systems are currently not at a level where they render more traditional text-based approaches obsolete, nor are they likely to be for some time. Therefore, there is still a need for text-based systems, and most systems in use today are still of this more traditional type. Such systems are able to retrieve well-annotated images with a good degree of precision. They are also able to annotate many images based on text surrounding those images. As long as the image descriptions are relatively accurate, this approach works well. However, images that are part of larger sets where the images are not annotated individually can be harder to retrieve, as whatever information is available for the set might not apply to all the images. Generally, systems will prefer to focus on retrieving images with better descriptions.

1.4.2 Multiple sources

Using multiple description sources for getting information about a single piece of data is not a new concept. The major web search engines use information about the different hyperlinks leading to a given site to determine both the contents and the popularity of the web site in question. In that case, each link functions as a different information source. However, aside from hyperlinks, most types of data only seldom have multiple sources describing their contents, and so most standard information retrieval systems focus on accurate retrieval based on a single description for each piece of information they index.

1.5 Contribution

This project has provided an implementation of the proposed idea of combining image collection context information to search for images. In doing so, it has first of all shown that this concept is feasible and is possible to implement. The system is able to search for images, and is able to combine collection information in order to provide an enhanced view of image semantics for the search process. Some of the challenges of making system such as this have been identified and described. Solutions to several of these problems have been proposed and implemented. However, the system implemented here is quite primitive, and there are a lot of venues for future work. For those problems where adequate solutions have not been created, possible approaches based on the work done here have been described.

Chapter 2

Background

2.1 Image search

The challenges of searching images are in some ways similar to those experienced in other fields of non-textual search. However, there are problems and features unique to the field. A brief overview of the most important approaches in image search is given below.

2.1.1 Semantic Gap problem

One of the main problems of image retrieval is matching the image or type of image desired by the user to an image available in the retrieval system. This is typically done by trying to deduce the actual meaning of a query given by the user, and matching this meaning to image information. However, as an unprocessed digital image contains no available comparable meaning, such a process relies on information either being manually added to or automatically extracted from the image. This, in turn, usually means that images usually have a very limited subset of the semantics contained in the image exposed, as manually adding such information is time-consuming, and automatically extracting it is very difficult with current technology. In an image retrieval context, the semantic gap refers to the divide between the semantics of a user's query and the available image semantics, the latter often being inadequate. [12]

2.1.2 Image retrieval methods

Research focusing on bridging the semantic gap problem described previously is divided in two broad categories, reflecting the two different ways of adding information mentioned above. The first category, extracting image semantics from the image data automatically through various techniques, is usually referred to as content-based, and retrieving images in this way is known as Content-Based Image Retrieval (CBIR). The second focuses on various ways of adding mainly textual information to the image, exposing its semantics by adding data derived from some form of manual processing. This can either be done directly (e.g. by tagging or describing images) or indirectly, by processing information already associated with the image, like usage context information, existing tags or descriptions, or other metadata already associated with the image. Retrieving images using textual information is known as Text-Based Image Retrieval (TBIR). These retrieval methods are not mutually exclusive, of course; newer research often tries to combine techniques from both approaches. [14]

Content-Based Image Retrieval

While much work has gone into developing various content-based techniques, and progress has been made in many categories (for instance facial recognition), the difficulty in creating a general system for the automated processing of image content is, if nothing else, a testament to the sophistication of the human visual subsystems. While many algorithms now are able to identify specific features like shapes and colours quite well, CBIR is still in many ways not mature technology ready to be used alone in general systems. Over time, there has been significant growth in research on CBIR systems, and such technology will likely become more and more applicable for real-world systems[4]. At present, however, the available techniques are not powerful enough to be used alone in a system for general image recognition with any sort of real reliability or specificity, nor are they easy to implement, as a large number of different complex systems would likely have to be used.

With all the research focused on it, CBIR is making progress, of course, and is becoming increasingly mainstream. Google Image Search[1] recently added some CBIR technology to their service, allowing users to specify different image types, such as black-and-white photos, clip-art, or images containing faces. A large number of experimental image search engines using CBIR techniques are available on the web, but none seem to really have taken off. CBIR can now identify many different general features with good success, such as faces, buildings flowers, and other quite sophisticated shapes. However, a big challenge is recognizing specific things based on known images, such as recognizing whose face or what building is depicted, based on other image of those objects. For the foreseeable future, therefore, CBIR will likely only be able to complement more traditional techniques.

Text-Based Image Retrieval

Those more traditional techniques are contained in Text-Based Image Retrieval, currently the method used in almost all general-purpose image retrieval systems today. This method uses the text associated with an image to try to determine what the image contains. This text can be text surrounding the image, the image's filename, a hyperlink leading to the image, an annotation to the image, or any other piece of text that can be associated with the image. This is largely an extension of the approaches used with other forms of text search. Google Image Search is a good example of a system using this approach to good effect.

There are two main weaknesses to this approach. The first is that one cannot use the information contained in the image itself; the information one is actually interested in is not processable. Instead, humans have to manually convert the image semantics to text in some fashion, requiring time and effort. While this will be done for some images when they are created or used anyway, this is far from true for all images. Second, one has to rely on the accuracy of such descriptions. Since only a minority of images are annotated expressively to give them an accurate description, one often has to rely on surrounding context information, which may not always be accurate or even related to the image semantics at all.

2.2 Collections & context

2.2.1 Image collections

In the literature, the term 'image collections' in many cases refers to large image databases, often well sorted and tagged and containing thousands of different images, as seen in e.g. [13]. In other contexts,

[1]http://images.google.com/

they refer to personal image collections or similar. They way the term is used usually changes depending on what setting it is applied to, often without an attempt to really specify what it means. The term is, in other words, not a well-defined one. This project uses a specific definition of image collection, where a collection is a set of images linked to a context. This definition is discussed in greater detail in the next chapter. This issue is mentioned here merely to make it clear that 'image collections' in the literature does not refer to the same concept used here.

2.2.2 Context

In an information retrieval setting, the term 'context' generally refers to information surrounding a specific piece of data. A more complete definition is this commonly-cited one, given by Anind Dey and Gregory Abowd: "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." [5] In this project, the entities in question are image collections, and the contexts used are of a specific type.

2.2.3 Usage context

The CAIM project has specified several distinct types of contexts as part of an attempt to better classify and describe individual image contexts. One of these context types, the usage context, is used extensively in this project. It is described as follows: "Usage Context represents information about the environment in which an image is used, for instance in an image collection or in a document where the image is used as illustration. Context information can be a textual description of a collection, a document abstract, keywords describing the collection or document, or text surrounding an image."[10]

The idea behind separating contexts into groups is that after they are classified, they can be described in a relatively standardized manner. Usage contexts can take many different forms, as the description above makes clear, but as a group they nevertheless often display some of the same general features, e.g. many will have titles, contain some text, be written in a specific language and have a specific source. If the most interesting of these features can be described, it should be possible to arrive at a reasonably standardized way of describing usage contexts. Any one usage context will likely not have all the possible features, but as long as the description can be somewhat flexible, omissions can be accommodated.[10]

2.3 Information retrieval

Techniques for retrieving information is a subject where vast amounts of work has ben performed. The majority of this work relates to textual information in one way or another. Advances in this field are still being made, but text-based techniques are now fairly stable technology. Therefore, text-based approaches form the basis of most conventional information retrieval today. Given the vast size of this field, and the large number of different algorithms that can be used, a full overview of this field will not be given here. While this project uses a few information retrieval concepts directly, it mainly relies on existing systems to do most of the underlying information retrieval work, and so the specifics of different retrieval approaches are not very important to the system's design.

2.3.1 Testing information retrieval systems

There are several different aspects of information retrieval systems that can be tested. The first level of testing is the purely functional examination, common to most software system. This consists of verifying that the system works; that it processes queries and retrieves results in the manner the design specifies. Once the system has been demonstrated to work correctly, one can proceed to measure the performance of the system. Of primary interest here is the response time, the time the system takes to process a query and deliver results to the user. The total time used, as well as the efficiency of various subsystems, can be measured. Another performance metric that can be measured is the storage space the system requires, if that is a limiting factor.

Lastly, one can test the system's ability to retrieve relevant answers in response to queries. This measurement is called retrieval performance evaluation, and is a type of measurement specific to information retrieval systems. This is sometimes a less straightforward thing to measure than functionality and performance, as the ultimate arbiter of what a relevant response is must ultimately be a human. Therefore, this type of measurement will in some way have to rely on test data sets created by persons qualified to judge the relevance of the results. A system or algorithm's ability to match these human rankings determines the retrieval performance. [2]

2.3.2 Text mining

While for instance taking a random document and automatically generating a complete description and an accurate keyword list is likely some way off, progress is being made. Automated text mining systems can now categorize some types of documents with a reasonable degree of accuracy.[9] Also, advanced systems focusing on specific domains can now determine relationships between abstract objects within documents, showing a sophisticated ability to parse text under certain circumstances.[11] The capabilities emerging from this field is interesting primarily when considering the possibility of automatically generating context descriptions from usage context data. Further advances should serve to ease the creation of the data this system uses.

2.4 Ontologies

In [1], a definition of what constitutes an ontology is given as follows, first quoting Thomas Gruber: ”...”an ontology is an explicit specification of a conceptualisation”. In this context, a conceptualisation means an abstract model of some aspect of the world, taking the form of a definition of the properties of important concepts and relationships. An explicit specification means that the model should be specified in some unambigous language, making it amenable to processing by machines as well as humans.”

In other words, in the context of Computer Science, an ontology is a set of words attempting to map some part of reality by finding and describing the words and concepts that make up that reality, then establishing the relationships between those words and concepts. While many ontologies are be limited in scope, others attempt to cover all language. They are used for several different purposes. One of the more popular is to enhance information retrieval systems in various ways, by helping to create hierarchies and determining relationships between different words and concepts.

2.4.1 Ontology use in image search

Processing existing image (or other multimedia) descriptions using an ontology is an approach taken in several systems (e.g. the Faceted Category system shown in [13], or the system combining CBIR, WordNet and existing tags in [3]). Most of these focus primarily on better categorizing individual pieces of media from their descriptions by using the hierarchical structure of WordNet or similar ontologies. Other projects attempt to use ontologies in systems using CBIR methods. These projects use various CBIR techniques to recognize features and objects in various images, then use an ontology to categorize these features, adding semantics and/or creating a browsable hierarchy for the processed images.

2.4.2 WordNet

One of the most widely used, commonly supported and best developed ontologies is WordNet[2]. It is a common choice for information retrieval projects seeking to use ontologies. It has a comprehensive database, and the many different interfaces implemented for it means that it can be used in most environments. It is also relatively simple to use, even though it is a complex set of data. WordNet is not a domain-specific ontology, but focuses on the entire English language, and tries to map all meaningful relationships between distinct concepts, distinguishing between different types of relationships, essentially forming a hierarchical structure. This allows one to see how different words are related, based one the types of relationships between them, and how strongly they are related, based on the number and types of links one has to navigate to get from one word to the other.

An important distinction WordNet makes is one between words and word senses. A word by itself is just the word as it is written, without any associated meaning. However, each word will also have one or more word senses, different meanings it can take on depending on how it is used. These are defined by the synsets linked to each word. A synset is a set of synonyms making up one of a word's word senses. Relationships between words are defined through these synsets, so that one can differentiate between the relationships for each of a word's meanings. When looking at relationships between two words, then, one actually has to look at the relationships between the synsets for the two words. If more than one synset is available for one or both words, one then has to decide which of the relationships to use.

When looking at relations between noun word meanings in the WordNet hierarchy, two important concepts are *hyponyms* and *hypernyms*. Hyponyms are subordinate word senses; they are a more specific form of the word sense of the superordinate word sense of which they are a hyponym. Conversely, calling a word sense a hypernym of another word sense indicates that the first word sense is superordinate to the other; the first word sense is above the latter in the hierarchy. Many words are both hypernyms of some words and hyponyms of others, of course, so the terms are used depending on which word sense's relations one is currently examining. [6]

It is important to note that these kinds of relationships between synsets are the primary ones WordNet contains. There are many other possible relationships between words, for instance between a verb and its typical object (e.g. 'drive' and 'car'), or between an entity and a part of that entity (e.g. 'car' and 'wheel'). These relations are not covered by WordNet. This means that there are many relationships between words that will seem natural to humans that will not be discovered using WordNet. However, this shortcoming does not outweigh the many advantages, mentioned earlier, this ontology has in an information retrieval context.

[2]http://wordnet.princeton.edu/

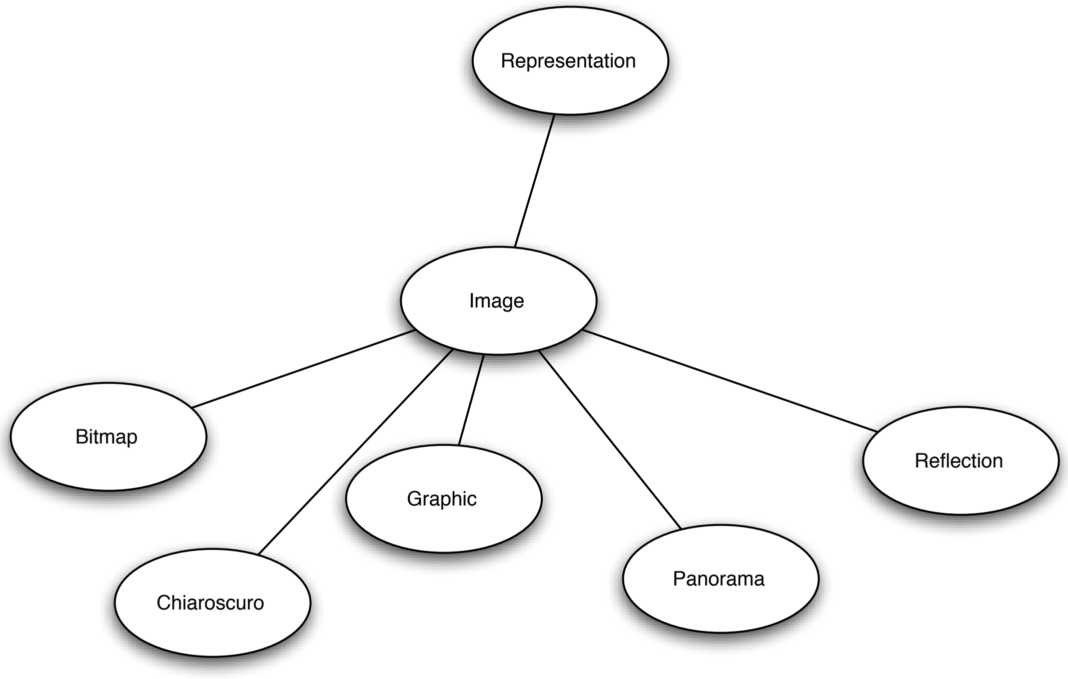


Figure 2.1: *Some of the WordNet relationships (hypernym above, hyponyms below) for the word 'image', used in the sense of 'a visual representation (of an object or scene or person or abstraction) produced on a surface'. Synonyms ('picture', 'icon', 'ikon') are not shown.*

Chapter 3

Design

3.1 Overview

The overarching purpose of this system is to use separate usage contexts for the same image to improve image search. This means that context information has to be somehow collected, searched and the results combined and presented to the user in a meaningful form. These three steps define the different design challenges for the system: collecting and storing the data, searching this data, and presenting the data. Each part has to be designed to with the later steps in mind; the purpose of storage and search is to allow the last part to function. The presentation part of the system is the only part actually displaying results for the end user; it is the only part of the system that shows the output that is the goal of this project.

3.1.1 Components

The system is divided into different components along functional and logical lines, as shown in figure 3.1. The divisions between these components structure both the system as implemented and the following chapter describing the system's design. Below are brief descriptions of each component. They are all described in more detail in the following sections of this chapter.

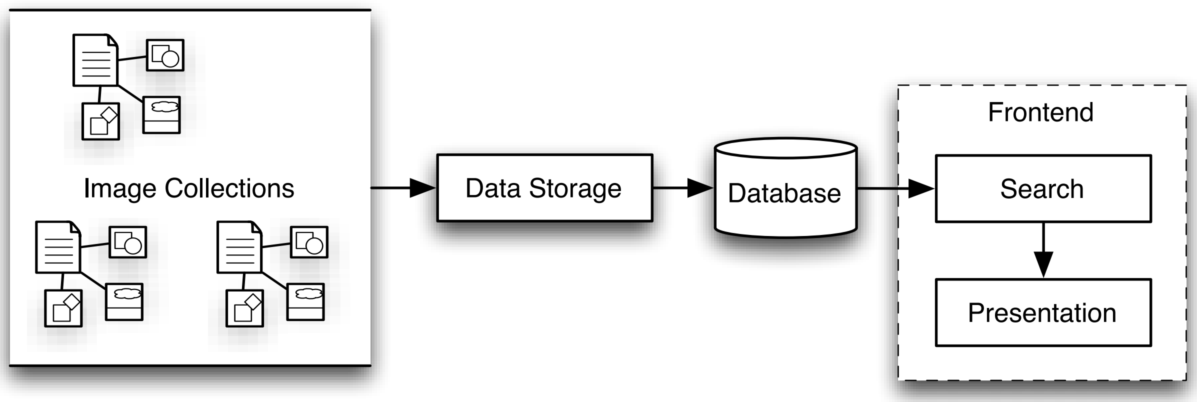


Figure 3.1: *Overview of system components*

Image Collections

These are the immediate external data sources, the data material that is processed, stored, and used as searchable information. Image collections have a strict form and definition in this system, and they are central to how this system works. They are therefore described in detail in the Image Collections section of this chapter.

Data Storage

Image collections are not used in their original forms, but are processed and stored in the system for later access. This component handles the initial processing and storage of the image collection data. This module is described in the Data Storage section.

Database

The database is the data repository where image collection information is stored and retrieved by the system. In design terms, it is not very complex, and so is described along with the previous module in the Data Storage section.

Front-end

The front-end of the system is what users will be interacting with when performing image searches. While its overarching principles are briefly described in a separate section, most of its workings is described separately when discussing the search and presentation components of which the front-end is made up.

Search

After image collections have been processed and stored, they are ready to be searched for, retrieved, and presented. This is handled by the front-end of the system. The search module is the first part of the front-end, processing queries and executing them to retrieve relevant information from the database. This module is described in the Search section.

Presentation

After a search has retrieved a set of collections in response to a query, the collections must be presented to the entity making the query request. The presentation module handles this task by examining the collection data retrieved and attempting to use this information rank the images in order of relevance to the query. This process is described in the Presentation section of this chapter.

3.2 Image collections

The term 'image collection' can mean several different things, as outlined in the last chapter, but is here used in a specific sense. For the purposes of this project, the term refers to a set of images linked to a single usage context. Both the design of and the reason for this project is intrinsically linked to the concept of collections. The basic idea of the project is to use collection information to get an improved understanding of the semantics of the images contained in them by combining collection information for those images found in several different collections. Image collections are therefore the basic unit of data which the system is designed to process. For the above reasons, they can safely be said to be the most important individual concept underlying the project.

3.2.1 Defining image collections

A collection is one or more unique images associated with one usage context. The source of a collection can have any number of forms; as long as a usage context description can somehow be generated from the source and images are clearly linked to it in some way, it is potential collection. In other words, as long as some form of textual information that can be turned into a usage context description is connected to a set of images, one will have a potential image collection usable by this system. A web page, a personal photo collection with a general description, a text processor document containing images are but a few examples of such potential collections. The usage context is always necessary, as it is the only part of the collection that actually describes its contents; a set of images on its own, with no usage context information, is of no interest, as the images can never be retrieved when using this system.

Creating a complete list of the different potential sources for collections will not be attempted here, as the definition is so wide. It is not really relevant here where a collection has its origin as long as the usage context and images are available. What kind of document or source a collection is based on does not matter when this system processes them; collections are by that point identical in their attributes and are therefore treated equally. The source document type might well be available in some form in the usage context description, but that is information relevant only to a user looking at collection data or searching the system. By itself, this information is not used by the system in any way. Collections, as used by this system can therefore be said to be homogenous; they share the same features, are all treated the same, and are described and stored in the same manner. While the source material may differ greatly, by the time the collections have been converted to simply a usage context description and a set of images, they are really not all that different.

Within a collection, the images are not differentiated in any way; they are all equally related to the context. This means there is no extra annotation of images based on surrounding text or image descriptions, for example. Nor is there any differentiation based on an image's features; its size, format and actual content is not of any interest for the purposes if this project except for purely implementation-related reasons. This does not prevent images being differentiated in other ways, by adding annotations to individual images, for instance, but such information is not seen as part of the collection information itself, but rather as ancillary data connected to the image.

A very important concept in this project is that the same image can be present in several different image collections. If an image in one collection is an identical copy of an image present in another collection, the images are for all intents and purposes the same. Since completely identical images will consist of the same digital image data, one can easily and accurately compare images and discover which are identical, without having to use any information other than what is found in the collections themselves. Being able to tie different usage contexts to one image allows one to combine the usage context information, and hopefully get a clearer idea of the semantics of the image.

3.2.2 Why use collections?

As collections describe their images through usage contexts, it is possible to use this information to search for the images that are part of these collections. But why choose collections as the foundation for an image search system when collections seem to offer less accurate information about individual images than e.g. tagging or other annotation of those images? The answer is that collections are there; they are available. While many images have been tagged or otherwise described individually, there are also large amounts of images that are not described in this way, and more are being generated all the time. Manually annotating these non-annotated images individually is not really a feasible solution. While some websites and projects have been set up for such a purpose (e.g. Google Image Labeler[1]), the limited amount of volunteer manpower compared to the task at hand means that one is unlikely to be able to annotate even a fraction of all such images.

However, these non-annotated images are often still used in documents, on web pages, or simply stored in personal or commercial image collections (in the general sense) of various types. They therefore usually have some information associated with them, information which can at least to some extent be assumed to also relate to the images the potential collection contains. This information is basically a by-product of the usage of the images in the collection source, and so requires no extra effort on the behalf of its creators. As long as this usage information can be extracted from the collection source and made into a description of the collection source, this information can be used to describe the images contained in the collection in a standardized manner. Even if the collection information is absent or inadequate, briefly describing a collection of several hundred images is obviously much quicker than annotating each image individually, although the result will also be less precise.

Since no better information images in such collections as described above is available, one would like to use usage context information to search among these images. The lack of individual information among the images can make this difficult, however, especially since collection information will often not be accurate for all images. Also, results from such searches can quickly become overwhelming, especially when collections become very large, and a user gets hundreds or thousands of images as a result of a search. Therefore, some sort of additional processing is needed in order to make such collections searchable in a useful manner, even when all collection information is available for searching. The approach chosen for this project is an attempt to make collection information more useful for search purposes, at least for a certain subset of images, namely those used in more than one collection.

3.2.3 Creating collection descriptions

Actually generating usage contexts automatically from text data is a task outside the project's scope. Therefore, this system does not handle the creation of collection descriptions from documents or other sources. However, the process that would be used is still of interest. When generating a collection, images would be extracted as they are, while the usage context would be generated, largely automatically, from the various types of textual information in the document. Many documents will have some of the more common interesting description information already available, such as the collection's title and its location. Depending of the type of collection, others may be present as well, such as short descriptions, keywords and the creator of the document.

One of the premises of this project is that usage contexts can be described in a fairly uniform manner. While the process of generating such descriptions is not defined, the process' expected results can still be specified. Some relatively standardized format will be needed for usage context descriptions to be treated in the homogenous manner this system intends. There are many different proposals for ways of describing semantic content. This project has been designed with one such new specification in mind.

[1]http://images.google.com/imagelabeler/

CTXT

CTXT is an XML-based format for describing image contexts currently being developed by the CAIM[2] project. It currently specifies some data fields common to all context information, as well as fields specific to usage and capture contexts. In this project, the CTXT format is used for formatting and storing the usage context information of image collections. The specification is still under development, and open to modification; this project uses the most recent example version available. CTXT currently specifies some general fields for all contexts, and then different fields for different subtypes of context. Only one type is used in this system, namely Usage Contexts. The example usage context specification given in (paper) is as follows:

Appendix A: Examples of CTXT vocabulary elements

Element	**What it means**
Elements of general usage:	
SourceId	The identification of the context source, for instance an URI.
SourceLocation	Identifies the location of the context source.
SourceDescription	A textual description of the context source.
Type	Specifies the type of the context.
Description	A full-text description of context information.
Language	The language of the context information.
Keywords	Descriptive terms specifically relevant for the context.
Specific elements for Usage Context information:	
Title	Title of the document or collection where the image occur.
Abstract	The abstract of a document.
Owner	The owner of the document or collection.
Creator	The creator of the document or collection.
Audience	Who the collection/document is intended for. (e.g. Children, Tourists, Students, Scientist)
Category	The theme of the collection/document. (e.g. History, Nature, Architecture, Sports, Travel, Culture)
Availability	Availability of the context. It can be public, private or restricted.

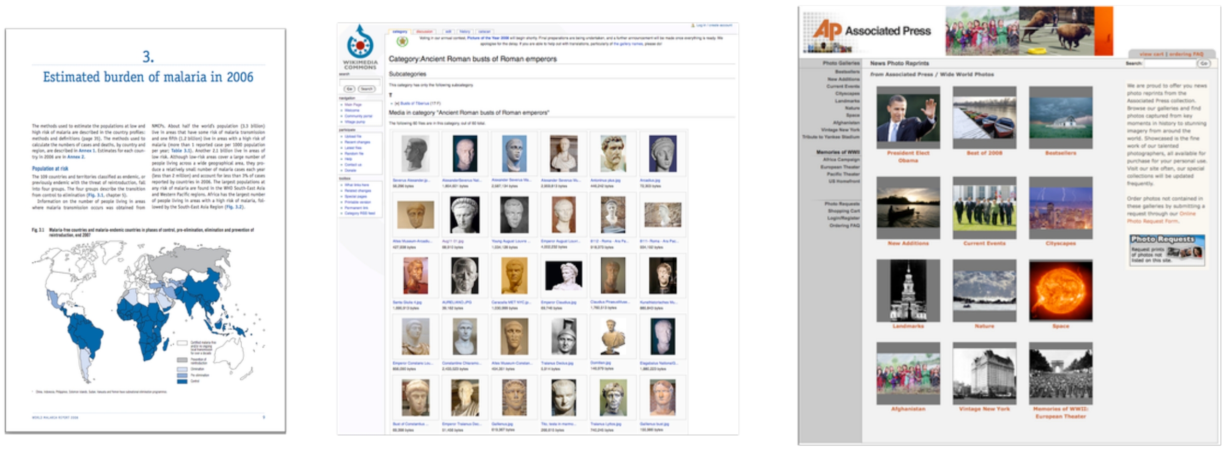
While the information required for all these fields will sometimes not be available for a given collection, there should usually be enough to generate at least a title, some keywords, and some source information. As long as the most general information is available, the context should still be usable as searchable data.

3.2.4 Collection sizes

The size of a collection is primarily measured by the number of images it contains, not by the length of the document or total size in bytes. Sources can contain a number of images ranging from the relatively small, like PDF files or similar documents, to the medium-sized, like web pages with tens of images or annotated personal photo collections, to the very large, like commercial photo databases containing several thousand images. There is no subject or type of collection source this system is specifically intended for, so as few assumptions as possible about what the typical collection source will look like have been made.

For the above reasons, there no clear conceptual limit for the number of images that can make up a collection, and there is no limit imposed in the system. Since collections can differ greatly in size, and there is no way to know what sort of collections will generally be entered into the system, some assumptions have been made for the sake of convenience. The expectation is that a collection will only

[2] http://caim.uib.no/


3\.

Estimated burden of malaria in 2006

Category:Ancient Roman busts of Roman emperors

AP Associated Press

Figure 3.2: *Example collection sources, from small to large: A document with images, a web page containing around thirty images total, a commercial image database*

rarely contain more than 1000 images, and that most collections will be made up of less than fifty images. The number of very large image collections for any one subject is assumed to be limited in comparison to smaller ones, and a further assumption is therefore that several smaller collections will be added for each large one. However, beyond possible search performance impacts, there is no reason why, for instance, only very large collections of 1000+ images could not be stored.

3.2.5 Limitations

The most obvious limitation of image collections as they are used here is that they provide no solution for having semantic information for individual images; all such information must be stored for the collection in its entirety. Indeed, when such information is available in an accurate form, it is usually superior to what this system can provide. However, there is no reason why collection information cannot be combined with information for individual images, and were the techniques attempted in this project to be put to actual use, they would almost certainly be combined with both individual image information and other techniques for indicating the semantics of images. In other words, this is a limitation that can easily be compensated for by using other, already existing approaches to the semantic gap problem.

Another challenge is that there is, unfortunately, no simple way to automatically measure how well a collection's images matches its context description. There is always a danger of having collections with a large number of images that are not related to the collection's context description. This is a problem shared by all Text-Based Image Retrieval techniques to some extent, but since a wrong or misleading usage context can here potentially give the wrong information for a large number of images, the problem can potentially be more severe here. Avoiding such collections must involve manually reviewing collections at some point. One possibility that has been considered is manually reviewing each context and assigning some sort of reliability score to it, describing approximately how well the images contained in it relate to the context information. However, this is time-consuming, and introduces the sort of large-scale manual intervention this system is meant to avoid.

An alternative is to introduce a system for rating the reliability of context sources. This would assign a rating to each collection based on where it originated and/or who provided it, giving some indication of the collection's reliability without having to examine each collection in detail. Several factors makes such an approach unsuitable for this system, however. First, it makes the assumption that a significant number of collections would be provided by the same source. This would likely be true for some situations where a system such as this would be employed, but far from all of them. Second, it still involves some level of human intervention. In the end, as the system has not seen real-world use, there is no way to conclusively know if such mechanisms are needed or not, and so leaving room for them to be included later is seen as sufficient.

3.3 Data Storage

3.3.1 Requirements

The basic requirements of this part of the system are that it has to somehow store or otherwise allow access to all usage context information and images to be searched by the system, and it has to allow for such searching to take place. As long as these capabilities are in place, the basic functionality of the rest of the system could be implemented. However, this requirement is ultimately too vague to be useful when building the system, and there are additional properties which, while not necessary, are desirable. First, ease of use and suitability when implementing is an important factor. Since the data storage itself is not really a central aspect of of this project, interesting experimental approaches are ultimately less interesting than something that can be set up and used quickly and reliably.

It is also important that searches in context data and associated relationships are accurate and responsive, retrieving all the information relevant to the query in a timely manner. Since search systems usually take requests from users who expect results back virtually instantaneously, introducing features that would add significantly (meaning several seconds or more) to the response time is not acceptable. Also, searches have to be consistent; with the same searchable data available and the same query, the system should return identical results. A user needs to be able to repeat a search as desired and get the same results each time, for instance. Finally, for each query, the system needs to examine all data that could be relevant, so that no possible results that could be of interest to the user are left out from the results.

In real-life usage, the system would potentially have to store a large amount of data, and also have to rapidly respond to a large number of queries, efficient data storage and responsiveness would also be important. But as this project is basically experimental in nature, and not intended for actual large-scale use, this has not been prioritized for the actual implementation of the system. As long as the system's design can support or easily be converted to support a data storage model which is efficient and responsive under the conditions described above, this requirement would be counted as fulfilled. Such a model would entail a high-performance, well-indexed database for storing context information, as well as relationships between contexts and images, likely specifically optimized for this purpose. Image information could be stored either in the same database or separately, along with the corresponding image data, which, because of the aggregate size of so many images, would have to be stored in another fashion.

3.3.2 Data storage solution alternatives

Design-wise, there are two main approaches possible when it comes to resolving the above-mentioned challenges. One can either go for a decentralized solution, where some of the information is not stored locally, or a centralized model, where all information is stored in one place. As mentioned above, this is largely a performance-related issue, and so this decision is not really important when it comes to achieving the primary goal of this project. However, down the line, when putting the techniques used in this system into practice, the consequences of this choice would be more important. Therefore, examining the options now lets one consider the possible requirements of each solution, hopefully allowing one to design a system which does not prevent either storage model from being used in future systems.

Decentralized storage

One could easily conceive of decentralized models where only the context information was stored locally for searching, and most of the image and collection data were represented by pointers to external resources, like URIs or similar. As long as there is information in the system showing which images are linked to which usage contexts, searching the context information is enough. This avoids duplicate storage of information already available, which reduces the need for storage resources drastically. One can also pass off the responsibility of presenting the data to the collection sources, rather than having to display images oneself. Most images are far larger than their associated information, and require significant storage space. In a larger system, indexing hundreds of thousands of images, storing all images locally would require significant resources for relatively little gain. The decentralized model here has a major advantage, as the need for storage would be quite low compared to a centralized model.

The main problem is that one then relies on the third parties that are actually storing the image and context data to make it available in a timely manner when it is needed. Because of the widely varying nature of the potential sources of image collections, this is not always a safe assumption to make. Whether one is dealing with with a general domain, like images freely available over the internet, or a restricted one, like indexing documents on an internal network, the image collection sources can for any number of reasons disappear at any time. To avoid presenting an unavailable image as a result of a query, one would then constantly have to monitor the collection sources the system is indexing. Furthermore, this requires collections to be based on sources that both the system and those making the query have online access to. In some cases, where all indexed data is accessible over the Internet, for instance, such a solution would not encounter any problems in this regard. However, if one wishes to index and present images not directly accessible in this manner, one has to implement some sort of central repository, essentially abandoning a decentralized model.

Centralized storage

A highly centralized model, storing all context and image data in a single database, was ultimately chosen. While some decentralized model would perhaps have been more elegant, requiring less resources and creating a more direct link to the source material for a query result, it would also be more difficult to implement, without ultimately addressing the objective of this project. It would also require a more complex set of test data; rather than just creating simple collections of images and associated context descriptions, the test collections would have to be more like actual documents.

A central data repository needs no real innovative data storage or retrieval techniques; a standard database would be enough to contain the necessary information. The usage contexts are stored in something close to their original forms as CTXT documents, using the defined CTXT fields as fixed rows in the database. This allows for easy translation between the CTXT and database storage modes, and allows for use of the CTXT format without having to interact with individual files during search. If desired, images can also be stored as binary data in the database along with their related information, to allow for ease of access later.

As long as the image data itself is not required during the search process, but is only used when storing collections and when presenting results to the user, converting the system to a decentralized model later should not pose a great obstacle. The centralized model is needed only when the system requires continual, rapid access to the collections themselves, rather than the usage context descriptions and image information derived from those collections. Due to the uncertain performance of the decentralized model, this is something a centralized model, where the data storage is completely controlled by the system itself, can deliver.

3.3.3 Storing collections

Image collections consist of two different data types: Context information and image data. Each is used for different purposes. Image data is used during the data storage process itself to determine which images are shared between collections. Images are also used after a search has been run, and the results are to be presented to the user; then, the image data itself is returned. Collection information is the data searches are actually performed on, and determine which images will be returned for a search. As the two data types are different both in nature and usage, they can be stored as separately as desired, as long as the links between them are preserved in some fashion. Also, since a collection only consists of these two data types, the collection itself does not need to be stored as a separate data type. By storing images, context information, and the relations between images and context information, one has in effect stored the collection itself.

3.3.4 Storing usage context information

Since a standard relational database is to be used for storing all data, and since usage context information is already present in the standardized CTXT format when part of a collection, the simplest solution is to directly store the context information relatively unchanged. For the purposes of this system, the most important CTXT data is can easily stored in a single database table, with each standard CTXT attribute being given its own field. Aside from the ease of the storage process, the most important advantage of doing this is that one can quickly easily and accurately access each field as desired later, since relational databases excel at just these types of operations. This is, of course, primarily important when searches are executed and quick, easy and accurate access is exactly what is required of the data storage system, as specified in the requirements.

Storing extended XML data

One problem with solution outlined above is the lack of any simple way of adding to the types data stored. Since the CTXT format is meant to be extensible, a usage context description can potentially contain any number of fields not part of the original CTXT specification. There are no really good ways of handling this in a relational database. Adding new columns to the table as needed is both inelegant and unworkable over time. Having a number of columns whose contents can be defined at time of data entry for each context is also inelegant, and with only a limited number one still has a theoretical possibility of running out of space. A third possible solution is adding a separate table, with one row for each separate extended field. Lastly, one can just dump all the XML for any extended fields into a single separate row in each context entry, then retrieve it if needed. It is this last solution which is used here.

This would not be a satisfactory solution for a system primarily concerned with the best possible storage of CTXT data, but is sufficient for this project. While this solution has several weaknesses, good handling of extended XML data is not strictly seen as necessary, seeing as the system does not really concern itself with this to any degree. Since this project is concerned with searching for and correlating different values for the same usage context description fields, extended XML data is not very interesting. Any one such non-standard field is unlikely to be widely used, and so the ability to easily search for it would not improve search performance to any great extent.

3.3.5 Storing images

From a design perspective, storing the image data is by itself rather trivial; simply storing it in the database works well enough for the system at this stage. For a non-experimental system, a more advanced, separate system of storage would likely be required, in order to reduce the burden on the primary database and to speed up image request response times. Since image data is generally far larger than even extensive context information, the resources required to store and serve images are far larger than for context information. However, storing everything in the same database is a simplifying measure with no real drawbacks at the experimental stage, and is in any case a decision easy to change later.

Identifying identical images

The most interesting procedure directly related to image storage is to find which images are identical, both to avoid storing duplicate images and to determine which images are shared by which collections. The most important consideration here is being able to do this reliably, preferably while not using too many resources. Reliably identifying identical images means both ensuring that images identified by the system as identical, actually are identical, while also making sure identical images are always identified as such. Currently, identical images are defined to be images containing the exact same image data, that is, the binary data contained in their image files is completely identical. A bit-by-bit comparison can establish this, but is a relatively expensive operation, as all new images will have to be compared to all existing images.

A better method is computing some hash value for each image and storing it along with other image data. Since using a given hash algorithm on the same string of bits will return the same hash value every time, one can be certain that images with different hash values contain different binary data, and so are not identical. Most hash algorithms are specifically designed to produce few collisions, that is, to provide unique hash values for different strings of data. If one uses an algorithm with a low collision frequency, it is then very likely that two images with the same hash value are in fact the same. [8]

When a new image is to be stored, one can compare its hash value to the existing hash values, which is far quicker than comparing the image data itself. If the hash value of the new image is not found, the image has not been stored in the system before, and the image can be stored as a new and unique image. If the hash value is found to be the same as the one of an image already stored, it is likely the new image is in fact identical to another image already stored. To avoid misidentifications due to hash collisions, which are very unlikely, but possible, one can then compare the actual image data for the images to make absolutely sure the images are identical. If the images are found to be the same, one can then simply associate the existing image with the new collection as well as the collection or collections it was already in, instead of storing it again.

3.3.6 Determining semantic links

Determining similarity

While having collection information available and knowing which collections share which images is a good start, one can do more. By seeing if there are possible semantic relationships, or links and similarities between the meaning of selected words, between the usage context descriptions of collections that share images, one can try to get a better idea of the actual content of those images. One way of exploring semantic relationships is to use an ontology. This leads is where WordNet is used in this system. WordNet can determine the distance between different two word senses in its hierarchy, allowing one to estimate approximately how much two words are related. As interpreting the semantics of language is very complex, this is far from a perfect solution, but can at least give some indication of whether or not two collections cover related themes.

To do this during the search process, one can first retrieve context descriptions found by a search for the original query terms, and subsequently also examine other contexts sharing images with those already retrieved. One can then examine this second set of contexts for words similar to the original search terms. This is likely to often be more accurate than e.g. a simpler approach of generating extra queries for synonyms of the query words: Since the contexts retrieved are already linked to images deemed relevant to the original query, it is far less likely that one is retrieving irrelevant context data. There is still a danger of up-ranking irrelevant images, but as long as this method is used with care, a more common outcome should be to more reliably determine the content of images.

However, in practice, this process is far to time-consuming to be used. Since a single word can have several different meanings, and each context will have several words to be examined, one ends up having to determine the similarity of a large number of word-meanings when comparing two collections. Since this process is based on determining the distance between two word senses in the WordNet hierarchy, a relatively costly procedure time-wise, performing this procedure even for a limited number of words between two collections can take a second or more. Having to do it for several usage context description pairs would quickly introduce an unacceptably large delay.

Finding semantic links on data entry

For the reasons outlined above, using WordNet in this way must be done when data is entered into the system, where these delays are not an issue. Since this is an experimental system, one might also perform lookups when searches are executed but in such a way that they could be done using static data generated when the data is entered instead. The problem with this approach is that one might quickly generate an excessive amount of relations between different contexts if the linking operation is not carefully constrained. One certainly can't map the relations between all contexts, since the amount of data would increase exponentially as more contexts are added to the database, and the potential number of contexts is very large. One therefore has to determine which ontology-determined relationships between contexts are most useful for the system, and which of these are practical to implement. Usefulness is here primarily understood to be more relevant search results, while practicality will be reduced primarily by the amount of extra data generated.

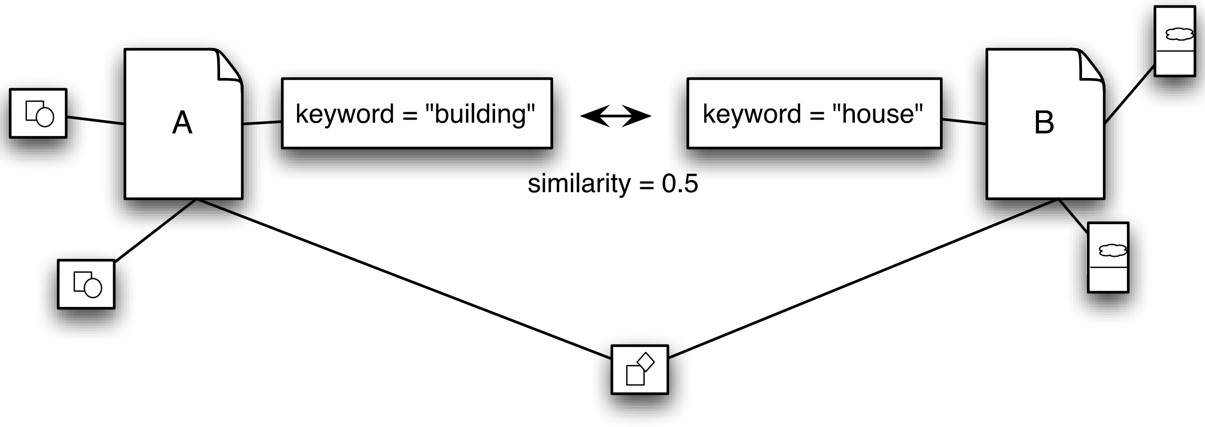


Figure 3.3: *Collections A and B share an image, causing them to be examined for semantic links. One keyword from A is rated as similar to one from B, which will cause a semantic link entry to be stored.*

Restricting number of links generated

In addition to the goals of usefulness and practicality described above, a secondary consideration is trying to avoid links between words that are lexically identical but semantically different; that is, words that sound the same but mean different things in their respective contexts. Conveniently, perhaps the best way of doing all this is limiting the establishment of semantic links to those contexts that also share images, as is already planned above. Since the system will mainly be exploring the similarities between contexts that share images anyway, and potential links between contexts that do not share images are not as interesting, this does not impact usefulness. However, it will greatly limit the potential number of semantic links between contexts to something approaching manageable. Finally, this limitation will also mean that one is less likely to be linking contexts which are actually dissimilar, as contexts sharing both the same words and the same images can be assumed to have an increased probability of semantic similarity.

One can then limit the establishment of these links still further by only storing those relationships whose strength is above a certain threshold. Weak relationships are unlikely to be as interesting as strong ones, and would in any case not have the same impact on final relevance computations as strong relationships. Weak relationships can also be assumed to be more numerous, as the number of possible relationships grows the more steps one takes in the WordNet hierarchy from one's origin point. Finally, one can restrict the number of relationship entries allowed between contexts to one, combining the information retrieved into a single relationship, losing some granularity but keeping the essential information. If necessary, one can then later use WordNet when searches are executed to determine the exact nature of those relationships. As long as one knows which words to examine, the costly operations needed to examine all possible relationships can be avoided.

An example of these principles in operation can be seen in figure 3.3. Two collections, A and B, share an image, and will therefore be checked for semantic links. When examining their keyword fields, one word in A's list, 'building', is found to have a best similarity score of 0.5 when compared to a word in B's list, 'house'. This similarity will be stored as a semantic link. This link represents the chance that an image present in both collections will have an increased likelihood of depicting whichever related words are found in both the collections. If the similarity score was lower, as for example between 'building' and 'car', a pair having the best similarity score of 0.1666, the words would likely be too unrelated to give any additional information about the likely contents of the image.

Of course, this operation is not performed for all fields. It must be restricted to those fields both deemed most important and containing relatively few words, as the process is still relatively time-consuming, and comparing two large bodies of text could take some time without more advanced methods in place to facilitate such comparisons. For now, only the keyword field is processed in this fashion, although the subject, category and title fields are also viable candidates for this process should it prove useful. One might also extend it to the description field; however, there is no fixed limit on the length of descriptions, and so these could of potentially any length, from just a short sentence to several paragraphs. Aside from requiring time-consuming processing, descriptions of different length would not really be directly comparable, as a longer description would likely generate far more connections than a short one, therefore automatically making it seem more relevant. There are ways to tackle this problem, but coming up with a good solution providing sensible results would require development time better spent elsewhere, and would in any case need the basic solution (keyword linking) to be tested first.

3.4 Front-end

3.4.1 Design goals

There are two main things the front-end has to do. First, it has to be able to combine context information in such a way as to get the best possible understanding of the contexts' shared semantics. This is where the combination of context information will increase the system's ability to show an understanding of a query beyond what simply using the context information uncombined would. Second, the system should use context information to provide more accurate results, prioritizing those images that are actually relevant for the user. Also, both of these goals have to be performed in a timely manner; ideally, a user inputting a query should have a response virtually instantaneously.

Performance is particularly important because relative to a basic information retrieval system, searching only in one table for individual data sources, a query in this system will require a large number of individual database searches and intermediate operations. This is because a lot of what is done in this system to improve searchable image information includes examining relationships between collections, rather than just displaying retrieved collection information directly. However, such examinations can often be time-consuming; examining all potentially interesting relationships might well take more time than one has available, and one must therefore be careful to not implement solutions which can never hope to be performed rapidly in practice.

3.4.2 Overview

The search and presentation components are, as outlined previously, both part of the front-end that handles the actual image search. While they are therefore closely related, and their boundaries are not entirely well-defined, they can still be separated design-wise by their responsibilities. The search component handles the first parts of the information retrieval process, which is to first parse the query, then execute it to find all context information stored in the database that might be relevant for the query. However, it does not do further processing of the results, and it does not determine which collections and which images are actually most relevant. The presentation component performs this task before finally presenting the query results.

A basic query where no images are shared between the contexts retrieved will function much like any

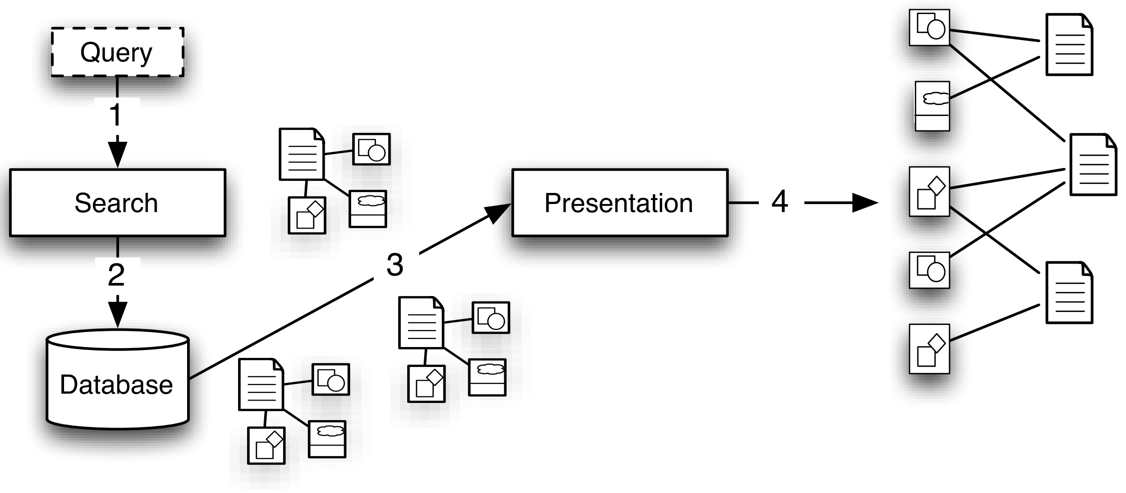


Figure 3.4: *The basic process of search and presentation: 1: A query is sent to and processed by the search module. 2: The search module executes the query in the database. 3: Collection information is sent to the presentation module. 4: The individual images in the collections are ranked based on collection information and presented to the user*

other document retrieval system: It will go through the database, picking out those contexts it finds relevant, rank them, then, based on these contexts and their ranks, display the images associated with them. As long as no images are shared, the system's advanced features will not come into play; indeed, since there is no provision for having data for individual images (unless they are added specifically as single-image collections), it is likely to underperform compared to systems based on such features, since it will be harder to find individual images that are a good response to the query. Instead, one has to return a large set of images and hope that those that are relevant will be picked out manually by the user.

3.5 Search

3.5.1 Query

Every search starts with a query, consisting of one or more strings and a set of options, specifying what results the user is interested in. The most important parts of a query string are the search terms, which indicate which words will determine if a usage context description is relevant to the query. The search terms are one or more words which will be matched to the values of those CTXT fields specified by the user. Before the query can be executed, it has to be changed from what the user put in to something that can be executed as one or more database queries. This is largely a technical matter covered later, in the Implementation chapter.

As it is hard to support all conceivable sorts of queries, support for a limited set of query options is used instead. These were selected both to cover most common query forms as well as to allow for those queries seen as most interesting for the project. Most are standard features found in modern search engines today.

3.5.2 Query Options

Stemming

As described in more detail later, the text search solution used features automatic stemming of search words, so all text-searchable data and all query words will be reduced to their word stem where possible (that is, endings like the final -s in a word's plural form removed). WordNet has a similar facility for doing the same, allowing words to be reduced to their stems where WordNet functionality is used as well. This allows a search for 'paintings' to return a hit when 'painting' is found, for instance. A downside is that there is currently no way to only search for a specific word form; one cannot search for 'paintings' without getting hits for 'painting' as well.

Boolean operators

Queries should support using the AND, OR and NOT operators to better specify the query. These operators are standard in most modern search engines, and together allow for a quite flexible way of entering queries. Per standard usage, the AND operator will mean that any search terms directly in front of or behind the AND must be a part of any result retrieved for the query. The OR operator will mean that any expression before or after the OR will be interchangeable when it comes to retrieving search results. Finally, the NOT operator will mean that no result including the search term directly after the NOT will be included in the result.

The definition of the exact behaviour of some of these operators is not a given in the context of this system, however. Since a single result (an image) can be based on several search hits for contexts, one has to decide whether the operator should be applied to each context separately, or collectively to all contexts related to one image. For instance, when searching for 'A AND B', should the system return only images linked to contexts that each contain both keywords A and B, or should one return images linked to a set of contexts where both A and B may be found as keywords, although not necessarily present in the same context? Similarly, should 'NOT A' remove from the results only those contexts where 'A' is found, or all those images linked to at least one context where 'A' is present?

The most restrictive behaviour has been decided on for the AND operator. Since there is no current support for grouping search terms (searching for "railway car" as a phrase, not as two separate words, for instance), some way of requiring several words to be present in a context seems to be in order. For the NOT operator, the least restrictive approach has been taken. Eliminating contexts from consideration will in effect lower the rank of the associated images relative to those images whose context descriptions are not removed. Instead of removing the images completely, they should still therefore be present if linked to collections without the term flagged by NOT, but ranked below those images linked to more collections without the term flagged by NOT.

Multiple search terms

Support for using multiple search terms without joining them through explicit Boolean operators is necessary. The primary difficulty with this is defining how exactly to the system should execute such queries: Should it simply treat multiple search terms as implicitly using the OR or AND operators, or should one treat such queries in an unique fashion? Generally, the trend in major search systems seems to be to require as little use of operators beyond the query terms themselves as possible, making it easier for end users to enter queries. This approach will be employed here as well.

For simplicity's sake, words do not explicitly need to be joined by Boolean operators, but any two words without any explicit operator between them will in effect be joined by the 'OR' operator when searching. However, it is important to note that this only applies for search purposes, and does not carry over to the ranking procedure. For example, while the query 'A B' will retrieve the same contexts as 'A OR B', making the same contexts available for ranking, the contexts can still be ranked differently depending on which query is used. A more in-depth description of this will be given in the presentation section.

Multiple field search

Most of the focus so far has implicitly been on keyword searching, but there are of course other context data fields defined by the CTXT standard available. Since these contain text in the same way as the keyword field, most can be searched in the same way, complete with Boolean operators and even using WordNet; for those fields where these methods are meaningless (for instance, the location field, which contains an URL), only relatively trivial extra functionality needs to be added. The important choice here is instead exactly how results for the different fields should interact, and how these fields should impact the final ranking.

3.5.3 Search execution

Finding relevant contexts

After a query has been input and processed for execution, the next step in executing the query is to retrieve all usage context descriptions deemed to be relevant. The ultimate goal of a search is to produce a ranked list of individual images, but as described earlier, individual images do not have individual descriptions of their content when stored in the database. All textual semantic information about images is connected to the collections they are a part of. To find relevant images, one must therefore first find relevant usage context descriptions. This means, in turn, that the first step is to determine which usage context descriptions can possibly be relevant for the query, so that they can be retrieved and more closely examined.

The first step in this process is to go through the database, examining the fields specified in the query for the search terms given. Any usage context description records matching the word or words specified by the query for a given field will be retrieved. Since the records retrieved can easily be checked against the query specification at later points, it is not important at this stage to differentiate between different results at this stage beyond keeping different records separate. What matters here is simply to retrieve all the relevant records in a way that allows them to be handled easily later.

Examining semantic links

The above process will return a set of zero or more contexts. After this first context set is retrieved, and there are one or more results, one can determine if there are any additional contexts semantically linked to contexts in the retrieved set; if so, their relevance to the query can be examined. To examine semantically linked contexts, one must first retrieve the information stored about the semantic links of all the contexts in the original result set. One can then examine the keywords the link consists of to see if they have a relation to the query. If one or more keywords in the semantic link are included in the query, the link can be said to be relevant. It will then be examined further to determine the similarity of the keyword matching the query in the collection already retrieved and the keywords in the linked collection, using the best match found as the similarity score.

Any such linked contexts deemed to be relevant will be stored as a separate set of related contexts, to easily distinguish them from the primary result set. Information on the strength of their semantic links must also be included, as these are used later in the ranking procedure. Any collections already present in the primary result set will not be retrieved here, as they will already be influencing the ranking of the images they contain, and including them in this result as well would in essence be to give them extra importance for no real reason.

For instance, say collection A from the previous example in figure 3.3 has been retrieved by a search for 'building'. All links that has collection A at either end will be then retrieved, because they are connected to A. The link will then be examined to see if it is relevant to the query. Since one of the words in that makes up the link is 'building', it will be deemed relevant and added to the semantic links in the result, with the similarity score of 0.5 as its strength. Unless collection B is also present in the search results, the strength of the link will be applied to the image shared by collections A and B.

Final steps

After all context information has been gathered, the image information for all collections involved will be retrieved, and the data retrieved can be passed on to the ranking system. The only image information needed at the ranking stage is the relations between images and collections; as previously mentioned, individual image information is not used in the ranking process. As the query is also needed for the ranking process when determining the relative relevance of different queries, it must also be made available in its original, unmodified form.

3.6 Presentation

In order for search results to be relevant for the user, they have to be returned in an ordered manner. A custom ranking system has to be used to process the search results after retrieval from the database. To do this a scoring system is used, assigning values to images based on a number of criteria. These values are largely arbitrary, in the sense that they are not based on any hard research or statistical data on what would be the best values, but rather on what intuition and experimentation has indicated to be reasonable. The specific values associated with the different ranking criteria are open to change; it is the ranking process and the reasoning behind the different factors that are included that has been deemed to be most interesting, as the values can easily be tweaked through further experimentation.

This ranking procedure is in one sense the central part of this project, as collection information has not been used to rank images before this point. However, most of the ground-work has already been done, and the process is therefore not very complicated. No new information pertaining to the ranking process needs to be retrieved from the database, and the actual process used is largely a matter of adding together different factors. It is therefore not hugely difficult to explain, as most of the concepts the process is based on have already been introduced in previous sections.

3.6.1 Ranking images

Each image's relevance will be based on the different collections it is part of, as no image by itself has any information usable to determine its own ranking. In effect, the rank of an image is a function of the rank of those its related contexts deemed relevant to the query. Rather than use a well-specified, formal ranking algorithm for doing this, the current more experimental, ad-hoc solution was created. The informal nature of the current process is not really an attractive design feature, as the lack of an overarching, well-defined approach to ranking can easily make the process unmanageable if new factors are added or if the ranking procedure is to be used alongside other ways to rank images, such as individual ranks for some selected images, for instance. The lack of a well-defined ranking scale, for instance, will make integration with other systems difficult, as coordinating

Each context's relevance, in turn, is computed using different criteria, depending on if the context was added to the results because of the main query or because it was found to be semantically linked to a collection in the main query results. If it is of the first type, it is ranked according to its direct relevance to the query and the number of images it contains. If it is semantically linked, its rating is based on the strength of the semantic link. Collections of the first type will have all their images added to the results presented to the user, while collections of the second type will have their rating applied only to those images included because the images are part of that first group.

Direct relevance

The most important factor in determining a context's relevance is how well it matches the query directly. If the terms in the query can all be found in the searched-for context field, the context is per definition fully relevant to the query. If only some terms are found, the context is still partially relevant, but will be ranked below any fully-relevant contexts found. In general, a context's relevance will increase in direct proportion to the number of query terms it contains; when searching for terms 'A B', a query containing both terms should be twice as relevant as one only containing one of them, other factors notwithstanding. The use of Boolean operators will of course modify the method of ranking; the OR operator will make search terms interchangeable, so that when searching for 'A OR B' a context containing both will not be more relevant than a context containing just one.

The direct relevance factor is seen as most important because it best reflects a collection's connection to the query given by the user. The text data in the usage context descriptions must be assumed to the best and most descriptive indicator of the content of the images. The other factors described below, while also interesting, should not be allow to overshadow this one when ranking an image. To further ensure such an outcome, this factor is additive for any given image; that is, if an image is a member of several collections all part of this primary result set, the ranking of all those collections are added to this image's final rating.

Number of images

At start of the query process, the number of images in a collection has no impact on how relevant the collection is for a given query. What matters is the relation between individual images and the information contained in the context description; a very large collection in many circumstances can contain fewer unrelated images than a relatively small one. For instance, a large image collection containing nothing but hundreds of generic images of commercial aircraft would be more likely to give a good match to the query 'aircraft' than for instance a small presentation of a specific aircraft, showing pictures of the aircraft along with images of interior details etc. However, the potential for unrelated images is assumed to grow as collections become larger, and so the system will slightly prefer smaller collections to larger ones when delivering results. A context containing many images will generally be considered to have a greater chance of containing images not directly relevant to its description.

It is arguable whether or not this is true; one might equally say that the nature of most large collections will be sets of images dedicated to a single theme of some sort, as in the previous example, and that there are few reasons to assume irrelevant images will be included; also, large collections with truly random contents will be quite rare. However, another perhaps more compelling argument for such a size modifier is that smaller collections will tend to be more specific, while larger collections will by their nature be more generic. Since a primary concern of the ranking system is showing the user the most relevant images first, showing the results from smaller collections before those from larger ones will often increase the chance of displaying the specific type of image the user is looking for. A context's ranking will therefore be very slightly reduced for each image it is associated with up to a set number; after that, no further reductions are made. This size modifier is applied last, as a multiplier to the ranking score previously obtained.

Semantic links

If a context that is deemed relevant has been determined to be semantically linked to another context sharing one or more of its images, this other context will already have been retrieved by the search module, and the precise nature of the contexts' relationship determined. If the context at the other end of the link is already included in the results, no further ranking adjustments are made. If it is not, and the semantic link is found to be relevant to the query, the second, related context is added to the results, and the link's relevance is applied to those images it shares with the first context only. In other words, no new images will be added to the results because a semantically linked collection is added to the ranking procedure. The increase in ranking from a linked context should not exceed the increase from a context actually matching the query. The use of linked contexts is primarily intended to give images that are otherwise ranked equal to others a slight nudge towards the top to reflect their slightly increased chance to be relevant to the user's query, and the increased likelihood of the images depicting what the context describes, due to the images being used in another, related context.

The linked context's relevance is primarily based on the semantic link's strength, in turn based on the distance in the WordNet hierarchy between the linked words. This is chosen as the most important measure because it is the best available representation of how similar the words are. While there are a number of sophisticated algorithms available for measuring the semantic similarity of words, the suitability of these would need to be tested, so a relatively conservative measure based exclusively on the number of links between words is used for now. In effect, this means that word senses with a relationship more than three or four WordNet connections distant will have a strength below the cut-off threshold used when first determining semantic links, as described in the data storage section.

The major problem with using this purely distance-based approach is that it is very sensitive to the somewhat arbitrary distances between concepts in WordNet. The link between 'vehicle' and 'automobile' would be below the cut-off threshold, for instance, because of the large number of concepts between them. While there are other, more sophisticated ways of measuring word similarity using WordNet, they all have their share of failings. The current approach is the simplest and also rather conservative, as it will tend to create relatively fewer semantic links due to the quick fall-off in similarity between words. It is in any case not very complicated to try out other algorithms later, should that prove of interest, but for now, the current approach is sufficient for testing the use of semantic links in this manner.

3.6.2 Presenting the results

The images that score highest at the end of the process are deemed to be most relevant, and presented as such to the user. As long as the final ranking is apparent and actual visual representations of the ranked images are readily available to the end user, the precise way this is done is mostly an implementation-related matter. The intent of the process is to end up with a list of images where those images that are part of several collections matching the query the query are at the top, those part of individual collections matching the query are in the middle, and those that are part of collections only partly matching the query at the end. The other factors, collection size and semantic links, should ideally serve to distinguish between images inside these broad categories.

Chapter 4

Implementation

4.1 Technology

4.1.1 Python

All original code for this project (excepting SQL database definitions) is written in the programming language Python[1]. The language was chosen mainly for reasons of convenience; some experience with it had already been attained, and it was known to be a language suited to quickly developing applications where creating a proof of concept was the goal and where the performance hit due to it being an interpreted language would not be significant. Also, it was known to be widely used and well supported, with a large amount of libraries and toolkits available for performing most tasks.

4.1.2 PostgreSQL

There are several different free, open source databases available. Having already settled on using a traditional relational database for data storage, and a large commercial database system like Oracle not being necessary for the current scope of the project, PostgreSQL[2] was more or less arbitrarily chosen over the other well-established open source database system, MySQL[3]. Postgres was chosen for a few reasons. First, some very limited previous experience with this system in operation was available. Second, Postgres' full-text search functionality, while somewhat more cumbersome than MySQL's, was also more flexible and therefore appeared more powerful. Configuring MySQL's full-text search would also in some cases require source code changes. Lastly, PostgreSQL is generally considered to be a more standards-compliant and robust system than MySQL; the latter lacks foreign key support for most of its storage engines, for instance. This sometimes comes at the expense of performance, but since MySQL's performance advantage was not relevant for this project, the more robust system seemed like the safer choice.

The psycopg[4] module is used for interfacing with the database. This module was chosen among the available Python database interfaces because its newest version is still being developed; most other Python database interfaces have not seen updates for some years, although several seem much more widely used than psycopg. The version of psycopg in use in this system is relatively old, as it is described as

[1]www.python.org
[2]www.postgresql.org
[3]www.mysql.com
[4]www.initd.org

more stable than the version currently in development, but upgrading to the newer version in the future once that is complete should be simple. psycopg implements the Python DBAPI v2.0[5] specification, which in effect functions as documentation for the module. It is relatively thread safe, allowing database connections (but not cursors) to be shared between threads, and so allows the application to run all database requests through one connection.

4.1.3 CherryPy

This relatively simple web framework for Python is used to create a user interface for the application. CherryPy[6] is very easy to set up, requires little extra code to be written in order to be integrated and run in an application. Applications using the language can be implemented using only standard Python and HTML code; no templating language or special functions need be mastered. It is a relatively simple system, having few advanced features and focusing mainly on just serving HTML pages as requested. However, it is a part of the more full-featured TurboGears[7] web framework, and so the system could likely be ported to this platform should more advanced features become necessary.

CherryPy works by mapping the URLs from HTTP requests to Python functions; a given URL will call a Python method by the same name as the last part of the URL. Any arguments included as part of the request will be passed to the method. The initial method called may in turn call other methods of the application, exactly as a regular Python application might. The return value from the method will be sent back as the result of the HTTP request. Ideally, the return value should therefore be well-formed HTML code or other data parseable by the application making the request. CherryPy can also be configured to serve static data, such as images or style sheets, thereby allowing the application to serve data such as images without having to rely on a separate web server or separate methods for handling requests for such data.

4.1.4 Python Imaging Library

The Python Imaging Library (PIL) contains modules for many different image manipulation operations, only a few of which are used in this system. The module is used primarily to generate thumbnail images of the full-size images stored in the database. Most image formats are supported by the module, but not all, so some more uncommon formats like Scalable Vector Graphics (SVG) images, used mainly for diagrams, are currently unsupported by the system. Non-standard images can be entered into the database as normal, but as thumbnail images cannot be generated for them they will not show up in search results. Implementing support for other image types would not be very difficult, however; functionality for generating thumbnails for any sort of image where this is possible could easily be tacked on later.

4.1.5 Natural Language Toolkit and WordNet

WordNet, by itself, is just a set of data, and needs an interface in order to access it. Several of these are available from the WordNet site itself along with the base WordNet distribution, but none allow for easy integration with Python. However, the Natural Language Toolkit[8] (NLTK) for Python, in addition to being an advanced system for natural language processing, contains an interface for WordNet. It is

[5]http://www.python.org/dev/peps/pep-0249/
[6]www.cherrypy.org
[7]turbogears.org
[8]www.nltk.org

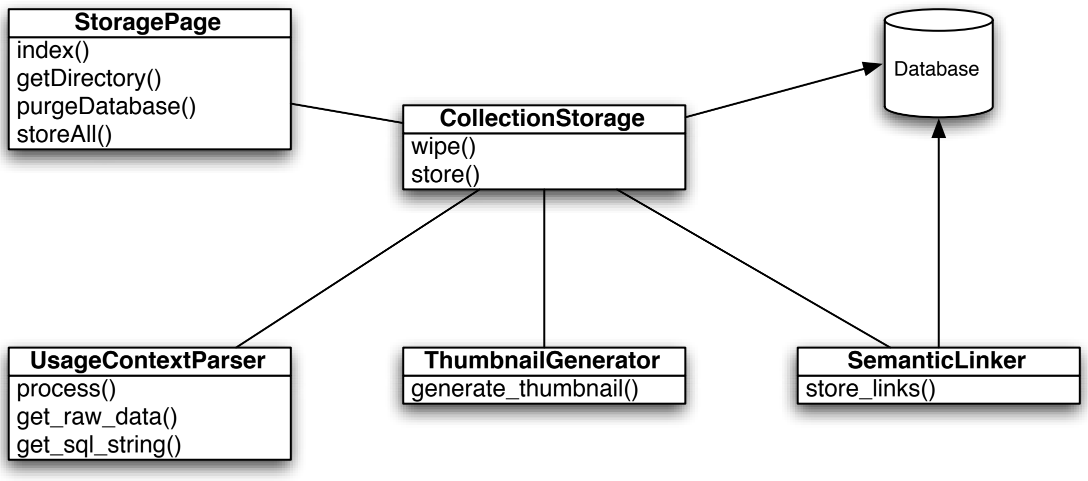


Figure 4.1: *Classes in the data storage module. The lines indicate which classes interact with each other. The arrows show which classes interact directly with the database.*

in the latter capacity it is used in this application; the many advanced features for natural language processing could potentially be used for better query processing and improved parsing of search terms, but such an endeavour would quickly go beyond the scope of this project.

The WordNet interface in NLTK allows for simple access to the various dictionaries that make up the foundation of the ontology. Once a desired word has been found, one can then get a list of the different synonyms separated into groups for the different meanings of the word, one can explore the various relations between the word and its neighbours in the WordNet hierarchy, and one can test for the distance between different meanings of the given word and another word, using various algorithms which measure the distance between the words using different criteria. Functionality supporting more complex operations is also included, but is not used in this application.

4.2 Data storage

4.2.1 Component structure

StoragePage: Simple web interface for controlling collection storage.

UsageContextParser: Responsible for parsing CTXT usage context description files in XML format, and rendering these into strings suitable for storing in the database.

CollectionStorage: Stores usage context descriptions and associated images in the database.

ThumbGenerator: Generates thumbnails of all stored images, so that these can quickly be displayed to the user when needed.

SemanticLinker: Detects and stores semantic links for newly stored collection.

4.2.2 Database

At the heart of the system is a PostgreSQL database, which stores all image and context data information. To store the data, the psycopg database interface module described above is used. The data structure is very simple, currently containing just three tables: One for images, one for contexts, and one to link images to contexts. The image and context tables each have a standard autoincrementing integer as primary key, while the context_image table has only two fields; one for an already existing context's key and one for the key of an image which is to be linked to the context. Since each image can only be linked to a context once, these values together function as the table's primary key.

Image table

As well as containing the image data itself, stored as binary data, the image table stores an MD5 hash of the image, used to rapidly compare image data, the image's original filename(s), the image's location on disk and a thumbnail version of the image, which can be used to rapidly display a preview of the image, useful when presenting all the images of a collection, for instance, where the sometimes very large image files would take too much time to retrieve. Image metadata values (like file size, type, dimensions etc.) are not currently stored in separate fields, as this is unnecessary in the current scope of the project, although a more fully developed version of this system probably would need to have this information more readily available. Another feature of this table which would probably be changed in a more complete version is the 'location' field. In the current implementation, this is used to ease the retrieval of the image data when displaying the full image in the web interface, allowing the image to be displayed directly using the disk address so that efficient image retrieval and display from the database does not have to be implemented.

Context table

The context table mainly contains the values specified in the Usage Context CTXT definition from (PAPER), with separate text fields for each value. In addition, it contains a field for full-text context storage, allowing for storage of the complete text of the context data. There is also a field for extended XML data, allowing for support of arbitrary extension of CTXT files. Almost all these fields are text fields, allowing PostgreSQL's text search functionality to be used by creating text search indexes containing processed text data for those fields where this is deemed necessary (mainly fields with longer, more complex text information, such as the keyword, description and full context storage fields.) Postgre's text search is the basic search functionality used in this system; while advanced queries are formulated in SQL at a higher level, the database itself can handle both the actual searching. Preprocessing these fields into text search indexes also allows for storing the fields in the indexes as a data type known as tsvectors in PostgreSQL, which are the results of preprocessing the text fields in preparation for searching. This preprocessing includes normalizing words according to dictionary rules (removing plural forms and other endings etc.), removing 'stop words' and some other operations. This is obviously a language-dependent process. Therefore, the standard language for the system has been set to English, and all searchable data entered has to be in this language for it to be parseable. It is theoretically possible to support multiple languages, but that is far beyond the scope of this project.

There are two other fields of note in the context table. The first is the extended XML field. While storing XML data in a simple text field is not an elegant solution, it is the only simple method that will support actually searching in these fields. PostgreSQL provides an XML data type, which stores correct

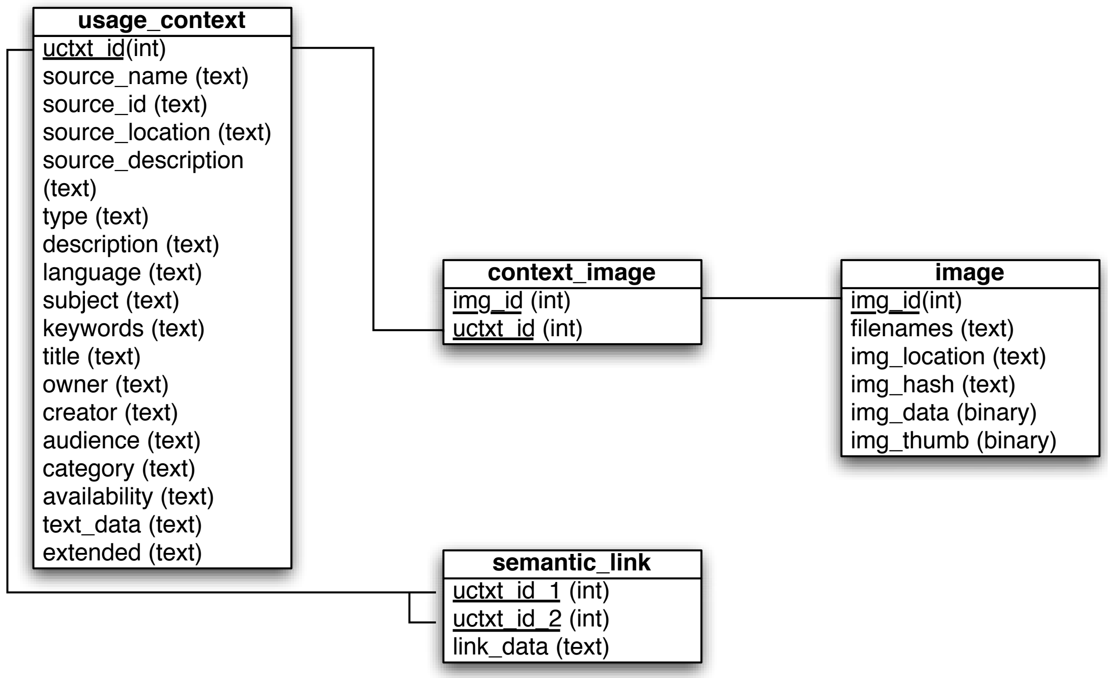


Figure 4.2: *Database tables and their columns*

XML documents, but fields of this type are not currently searchable, and there would then be little point in including this data in the current version of the system, which is not much concerned with retrieving data from the database for purposes other than searching. Other more complete solution would probably require significant time and effort to implement, for little real return.

The other field which should be mentioned is the SourceId CTXT value. This is treated as an unique identifier, which seems to be a reasonable assumption, as the CTXT specification states this may be an URI for public contexts. It is therefore classed as unique in the database, so that two contexts with the same identifier cannot be stored, something which must be considered when generating data for the database. Having an unique identifier not generated by the database prevents accidentally storing the same context twice, something which would likely impact search results. However, it could lead to problems in a real-world setting if one is not careful; for example, while the URL of a web page might be the same, the content might have changed. One then has to either discard the old data or find another way to separate the two entries if one wishes to retain the unique identifier-approach. This problem is discussed further in the Future Work section of the last chapter.

Semantic link table

Each entry here in the semantic link table is meant to hold all stored data for semantic links between two collections. In other words, any two collections will not have more than one entry between them. Therefore, the combination of their unique IDs is sufficient as a primary key. There is just one other field in this table, a text field where related keyword pairs and the strength of their relationships are stored. This allows multiple semantic relationships between two contexts to be stored without creating additional fields or haiving multiple rows for each collection pair. Since the keywords are not searched for in this table, storing them in such a relatively inaccessible manner does not really present a problem.

Physical database setup

The system has been set up to allow the database to run on a computer separate from the one running data entry and/or search operations. It is therefore also possible to run several different search processes on several different computers; since searches do not modify the database, these processes do not interfere, and do not need to be synchronized. However, it is not recommended to use more than one computer for the storage of data, as the sequence of operations involved in storing images could potentially cause the same images to be stored twice if two contexts containing the same image were to be stored at the same time. Since there is no provision for database synchronization, there is currently no way to easily use several different databases, although this would be relatively easy to implement as the data is not frequently updated. Such redundancy on the query or database server level might be a solution for the performance issues that have been mentioned as a possible problem for this system with its relatively high number of operations and database queries per search.

4.2.3 Adding data

Storing data in the database is for the most part a rather straight-forward process. Collections are initially found outside the database, on disk in separate directories, containing the images in the collections as well as an XML file containing the CTXT information for the collection. The store() method of the CollectionStorage class takes one such directory as an argument, storing all compatible data found. A simple web interface is used to add and remove data from the database, implemented using CherryPy. It is currently quite rudimentary, allowing only three operations: Storing a single collection, storing all collections, or removing all data from the database. It uses the store() and wipe() methods of the CollectionStorage class to perform these tasks, calling the store() method repeatedly for all the subdirectories of a config-specified path when asked to store all data.

Storing usage context descriptions

The CTXT XML file is first read and processed by the UsageContextParser class, which translates the XML value names to the names of the database fields. The XML is processed using the Python miniDOM parser included in the base Python distribution, making the process quite effortless as long as names of the values are known. It then prepares the SQL statement used to store the context data in the database. This string contains all the field names as well as their associated values. The context data is then stored and committed to the database using this SQL statement, allowing images to be linked to it.

Storing images

The images in the collection are then processed and stored separately. For each image, first the image data is read, and a MD5 hash of the image file is computed. Since two identical image files will have the same hash value, this hash can then be used to check if the image already has been stored as part of another collection. If the image's hash does not already exist in the image table, the exact image the image file contains is not present. The image is then stored in binary format along with the image's hash string and the image's filename. The image is then linked to the context information of the collection being processed using database primary key values for the respective tables.

As the MD5 algorithm is a cryptographic hash, it is designed to have a very probability of collisions, and so it is extremely unlikely that two images will generate the same hash value unless they are identical. Nevertheless, the possibility that different images will have the same hash value exists, and so if the hash value is found to be associated with another image already stored, the data of the two images will be compared to see if they match. If they do not, the image is stored as above. If an existing copy of the image is found, the filename value of the image is updated with the name of the new image file, and the existing image is linked to the newly added context information.

Finally, a thumbnail image for the file is generated by the ThumbnailGenerator class. The Python Imaging Library is used here to resize the image to an appropriate size using its thumbnail function. The new, smaller image is then saved as a .jpg file in a specific subdirectory, common to all thumbnail files, using the image's unique ID to distinguish it from others. Unfortunately, this process does not work on some more esoteric formats (like PDF vector images and SVG files), and so thumbnails are not currently generated for some non-standard image types. The thumbnail is used when presenting search results, allowing for smaller versions of the images in the result to be presented to the user for his perusal. The thumbnail image is added to the database, but also stored in a central location on disk. Generating the thumbnail images beforehand and storing them separately on disk allows the results to be displayed without using significant time to retrieve the full-size images from the database. As the files are typically very small, they do not require significant disk space in an experimental setting.

4.2.4 Semantic links

After all images are stored, the system will check to see if any of the images of the collection being processed were already stored in the database as part of other collections. If so, the detection of semantic links is initiated. First, the keywords are retrieved from the usage context information for all the collections sharing images with the newly stored collection (these older, related collections are here called 'existing collections' for simplicity). Since semantic linking is currently done only for keywords, this is all the information that is needed. This information, as well as the keywords for the collection being stored (or 'new collection' for short), is passed to an instance of the SemanticLinker class, which has access to a local copy of WordNet through the Natural Language Toolkit (NLTK) described previously.

The relations between the new collection and the existing collections sharing images with it is then examined, one collection from the related set at a time. First, all the keywords are reduced to their stems through a stemming algorithm built into WordNet, accessible through the NLTK. This causes plural forms and the like to be changed into the word's basic form, which is the only form recognized by WordNet. Then, the similarity between each keyword associated with the new collection and the existing collection is computed in turn. Each of these similarity-calculating operations requires further steps: First, each word is looked up in the four WordNet dictionaries, one each for nouns, verbs, adjectives and adverbs. Then, if the word is found in one or more of these dictionaries, all the word senses, or sets of different meanings a word can have, are retrieved from the different dictionaries. If the word is not found, it is simply skipped. Since one does not know which of these are the correct sense for the keyword in question, one must at the outset assume them to be all equally likely.

Similarity rating

Then, the distance rating between all of these meanings are calculated. This distance rating ranges from 1 to 0, where 1 is returned the word senses are identical and 0 is given when there is no connection between them, and the score goes from 0.5 and down if there are one or more steps between the word senses. The distance score is a rough reflection of the similarity between the two words senses, and so it is this score that is used to rate the strength of the relation between two keywords. However, as many, if not most, words have several different senses in which they can be used, one is therefore likely to end up

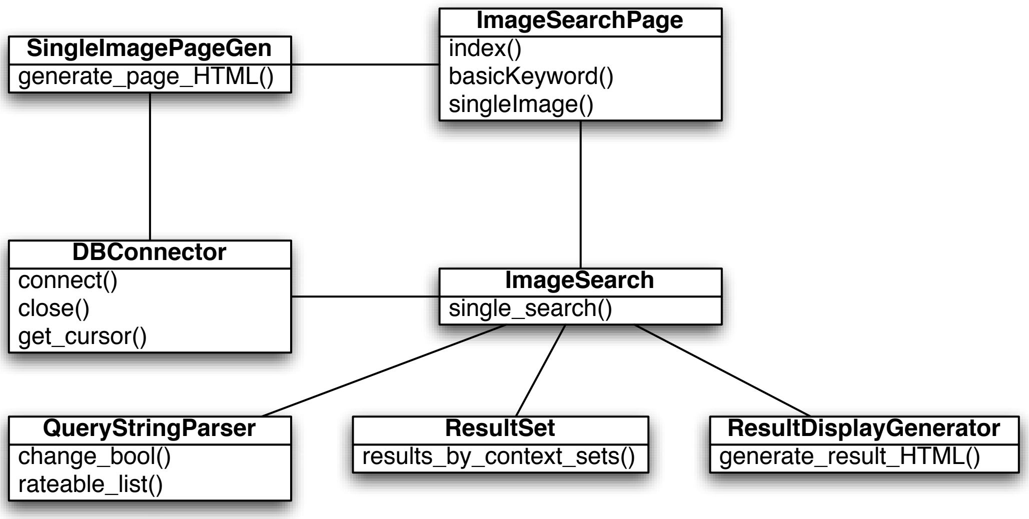


Figure 4.3: *Classes in the search and presentation modules. The lines indicate which classes interact with each other. Those classes connected to the DBConnector class are those who also interact with the database.*

with several different possible similarity scores for each keyword pair, one for each possible combination of word senses.

Since the collections share images, one can assume that there is a reasonably good chance that their usage contexts also touch on the same subjects. Therefore, a further assumption that can be made is that any two keywords having a closely related word sense are indeed used in the usage context descriptions with that word sense in mind; that is, the strongest similarity rating is more likely than the others to be correct. For this reason, the strength of the strongest link, rather than some averaging score of all possible similarities, is used when determining the true strength of the similarity between two keywords.

Storing links

However, as described in the design chapter, not all semantic links are strong enough to merit storing. Therefore, a cut-off value of 0.25 has been set. Any keyword pair with a best similarity below that value (representing a distance of three links) are not stored. All keyword pairs with a similarity equal to or above 0.25 are collected, and these pairs, along with their scores are added to a string. This string is then stored in the semantic link table along with the context IDs to which they belong, creating a compact record of the possible semantic relationships between their collections.

4.3 Front-end

4.3.1 Component structure

ImageSearchPage: Displays the web interface where the user enters the query, handles activation of other methods.

DBConnector: Utility class handling database connections for all classes querying the database.

Search

The three classes below take the query and uses it to retrieve information from the database.

QueryStringParser: Parses the query string given by the user, converting it into a format that can be used for searching the database.

ImageSearch: Handles searching for relevant usage context descriptions.

WordNetContextRelater: Examines semantic links, and retrieves linked collections.

Presentation

These classes process information retrieved by the classes above, then present it to the user.

ResultSet: Contains the results from the search process, and contains methods for ranking those results.

ResultDisplayGenerator: Generates HTML for displaying ranked results.

ResultPresentationPages: Generates HTML for a page containing one image in full size, along with some information about the collections the image is a part of.

4.3.2 Web interface

The front-end interface is displayed to the user as a web interface. The CherryPy web framework described previously is used to render the pages for this interface, using simple Python functions returning HTML, as described in the section on CherryPy above. As CherryPy includes its own simple web server, there is no need for separate server functionality. Although obviously not up to the performance standards of a dedicated web server like Apache, the CherryPy server is more than adequate for experimental purposes. As CherryPy is designed to easily use other web server solutions, moving to a better-performing web server should not prove much of an obstacle should the need arise.

Using CherryPy in this manner is extremely simple. The first step is to write a Python function returning the desired HTML for each page. Second, the function must be set to be exposed to CherryPy, allowing one to hide those functions one does not wish to serve up as web pages. Finally, one can set one function to serve as an index page, which will be the page presented when connecting to the CherryPy web server. In this system's case, the page generating the HTML form where the user is to specify a query is set as the index page. Connecting with a standard web browser to the computer running the search subsystem is all that is needed to bring up the search interface.

4.4 Search

In implementation terms, the Search module handles the part of the image search process from where the query is submitted by the user to where the relevant data has been retrieved from the database. Between those points, the bulk of the work is searching for and retrieving information from the database. There are three main steps to the search process: The first is to convert the query from the string submitted by the user into something that can be handled by the database. The second is running the text search operation and retrieving the results. The last step is to examine the search results, then retrieve any semantic links that might be relevant for the query.

4.4.1 Query

Entering a query is simple, and is fairly close to how it is done in standard web search engines like Google. The user selects the fields of interest, then types in a query string with the search terms he is interested in, using the Boolean operators AND, OR and NOT as described in the design section. When the query is submitted, the page will call the ImageSearch function with this string as an argument. PostgreSQL also uses Boolean operators for the query string, but uses different symbols for such operators. The string will therefore be converted to a statement suitable for use with the PostgreSQL text search function. Also, all single words must be joined by the AND or OR operators for a statement to be run correctly, so any words not joined explicitly by the user will be joined by the OR operator. This means that for the purposes of the database query, two words explicitly joined by the OR operator will be treated the same as two words entered by the user. However, the presentation module will later treat words joined by the OR operator differently.

4.4.2 Execution

Running the text search operation is done through a single SQL statement, which looks approximately like this:

```
SELECT uctxt_id
FROM usage_context
WHERE to_tsvector('english', <field>) @@ to_tsquery('english', <query>)"""
```

When run, <field> will be the field that is to be searched (like title, keywords, etc.) and <query> will be the query string. to_tsvector and to_tsquery are database functions converting the query and data in the database to a special text-search format, while the '@@' is the PostgreSQL text search operator. These functions are described in more detail below. The SQL statement involves several operations in the database. First, it will cause the query string and the data in the database to be converted to a format used by the text search engine. Then it will run the converted query on the data that has been prepared, searching through the text data in the database.

4.4.3 Text search data conversion

When the query has had database-compatible Boolean operators inserted, the string will be used in a PostgreSQL text search operation on the selected fields of the usage context table. First, a built-in database function, to_tsquery, will prepare the query string for the search. to_tsquery will start by by stemming all query words that can be stemmed. It will then remove all stop words, which are words like 'a' and 'the' which by themselves carry little information. Lastly, it will change the string into the tsvector data type, which is the data type used for text searches in PostgreSQL, both for the query string and the text in the database. Each tsvector contains a set of tokens, which are all unique stem-words in the converted string, excluding stop-words. It also contains information on each word's position in the text, so that a word occurring multiple times has only occurs once, but has multiple locations. These operations are currently restricted to the English language (the argument 'english' in the SQL example above indicates which language setup is to be used.)

Then, a text search index will be built for the fields of interest. The only thing needed to create this index is to give a command to the Postgres database, and it will handle the rest. The index creation function will transform the plain text strings found in the database into the tsvector data type using a process similar to the one described above for the query string. Creating the index in this way is obviously inefficient, as it has to be recreated each time a query is run. However, with only a limited set of collections present (20 or so) for testing purposes, there is no discernible delay when creating the index. It is possible to set up such an index permanently in the database, and there would only need to be some very minor changes to the SQL queries executing the text searches, but care must then be taken to update it whenever more data is added. Since the system currently performs well without adding a permanent index, it has not been deemed necessary, but it is likely that if a large number of collections are added and the system sees heavier use, this would have to be done.

Text search

After the query and the data is ready, the search will be run. The '@@' operator in the SQL statement above is, as mentioned earlier, the text search operator. This operator indicates that the query to the right of it should be run on the data to the left of it. This will produce a result set containing all usage context description entries whose values for the given field contain search terms corresponding to those given in the query. The search is run once for each field that is to be searched, giving one set of results per field. The results will contain the database ID of the usage context descriptions found, the titles of these usage contexts, and the data for the field being searched. These results are then merged into a single result set, removing any duplicates resulting from hits to more than one field for the same context description. The data for the fields being searched is included as a list, with one element for each field.

While there are a number of advanced options available for PostgreSQL text search queries, none of them are utilized here. In particular, there are some options available for ranking the results based on their contents. Among other things, this system allows for weighting of results based on which fields the query terms were found in. Using this ranking system would not cover all the needs of this project, but it could, at least, assign a rank to different usage context descriptions before further ranking was performed. However, as a ranking system had to be implemented later anyway, involving another ranking system did not seem to be necessary before the custom ranking system of this project had been implemented and tested. It would also tie this project to the PostgreSQL database platform without; in most regards, the PostgreSQL database could be switched with another relational database with text search support without much trouble.

4.4.4 Completing result set

After all usage context descriptions that match the query have been found and retrieved by the text search, image information for all these collections are retrieved. First, the image IDs linked to each of the different usage context descriptions are retrieved and stored as a list alongside their linked descriptions. Together, a context description and its associated image IDs have the information necessary describe a collection; actual image data is not necessary, as it can be retrieved later, and is not relevant to the ranking process. In this way, the data collected up to this point represents all collections directly relevant to the query to the extent necessary for processing by the presentation system.

The semantic links of the descriptions in the result set are then retrieved. Since the IDs of both context descriptions is given as part of the link entry, retrieving these semantic links are straightforward; one simply needs to search for all those context IDs that are part of the results described above. The link data itself is stored as a single string, and is not processed further here, but passed on to the presentation module along with information on which two contexts it links. For each link, those images shared by the two linked collections are also retrieved, so the ranking system later knows which images to apply the link's rating to.

Calculating number of search terms present

The text search will not give any feedback about which search terms are present in the query. Therefore, this has to be determined through a separate process. However, since both the query and the searched data is modified by the database through stemming and removal of stop words before the query is run, a comparison of the original query and the data retrieved will not yield results comparable to the text search made by the database. Both the retrieved data and the original query will be in their original, unstemmed forms, and will still contain stop words. The easiest way to deal with this is to use the to_tsvector function from the database on those words and fields that are to be compared. Therefore, the to_tsvector function will be run both on the query string and the values retrieved from the database for both collections and semantic links. The results can then be used by the presentation module to easily calculate the nmber of search terms present.

Creating ResultSet instance

All this information, usage context descriptions, their associated images, and the semantic links relating to the usage contexts, is loaded into an instance of the ResultSet class. This class, containing all the information gathered above, is what will be passed on to the presentation phase to provide data for the ranking process. The original query string is also added to the ResultSet instance, both to indicate which query generated these results and to make the query string available for the ranking process.

4.5 Presentation

The central part of the presentation module is the ranking process. This ranking operation is largely performed by the ResultSet class itself, using internal methods to perform calculations on the data it contains. The ranking operation will begin by ranking context descriptions separately. Then, for each context, its rank will be applied to those images linked to the context. After that, the value of any relevant semantic links will be applied. This process ultimately produces a ranked list of the images linked to those usage context descriptions found in the search process earlier. When this list is ready, it can be returned to the method in the web interface which requested that the query be made.

4.5.1 Determining individual collection ranks

When the ResultSet class is instantiated with the data described in the previous section, it will immediately calculate the ranks of all the collections part of the primary results. First, the usage context description's relevance to the query will be calculated, largely relying on matching words in the query to the data for each collection. Other methods can then be called to calculate the individual ranks of images and to get the the results of these calculations.

Relevance to query

Query relevance is calculated by seeing which of the query terms are found in the data for the usage context description in the result set. The base relevance score, given to those collections deemed to be maximally relevant, is 2. The number of search terms present in the collection has already been determined during the search phase. For each usage context description, the number of query words found in its data is compared to the number of words in the query. This will produce a number that is equal or less to 1, and greater than 0. This number is then applied to the base relevance score of 2, and the result of this calculation is the collection's relevance score.

Support for multiple fields is still rudimentary, as the focus of this project has been on using the keyword field for implementation and testing of various features, most notably semantic links. Fields are therefore treated as being the same with regard to their importance; that is, results for the title field are not more important than the keyword field, for instance. In essence, including more than one field in the query will combine the data from all the fields for ranking purposes. For instance, if searching in the title and keyword fields, the values for these two fields combined as one string will be what the query will be measured against.

Number of images

The value arrived at above will be modified based on the number of images that are part of the collection. The number of images will be multiplied by the largely arbitrarily chosen factor .001. The number arrived at will be subtracted from 1.001, leading to a number that will be 1 or lower. A lower bound for the collection size modifier has been set to 0.5, meaning that collections with 500 or more images will all have this multiplier, and the modifier will never become negative. The collection size modifier is then multiplied with the collection's relevance score, producing the collection's final rank.

4.5.2 Ranking images

After collections have been rated, individual images can be ranked and presented to the user. With all the information available at this stage, this is a fairly straightforward process. For each image, there are two factors that will determine the image's final score.

Usage context description

Each usage context description in the results is gone through in turn. Their relevance score is applied to each of those image IDs that is part of their collection. This works in a simple additive manner, so that an image that is part of two collections, each with a relevance score of 1, ends up with a score of 2. Images that are part of only one collection will end up with the score of just the collection they are part of, causing them all to end up with the same rank.

Semantic links

The strengths of any semantic links are then applied to those images in the result list that are part of both the collections in the link. This information has been retrieved during the search phase, and is part of the ResultSet object. The actual number added to the image's score is the link's strength divided by 2. This should result in a value of maximum 0.5, as the maximum strength of a semantic link is 1. This number is kept relatively low as to not overwhelm images that are part of the primary result set but are not part of collections with relevant semantic links.

4.5.3 Displaying results

Finally, the ResultSet instance will generate a ranked list of the images in the result. Ranks are not necessarily unique; as mentioned, images part of only one collection will end up with the same rank. This ranked list is then used by the web interface to display thumbnails of the images, which have been generated previously, during the data storage phase. The images will be displayed in order, with the highest ranked images first. The user can then click on individual thumbnails to have the full-size image displayed, along with some information about the collections the image is part of.

As the full-size image is stored in the database, it will first be retrieved, then stored to disk in order for it to be served by the web server. Images stored in this way will currently not be deleted, so first a check is run to see if the image is already available on disk, effectively functioning as a cache. The CherryPy built-in web server allows for easy serving of static data such as images, so both the thumbnails and the full-size images can be displayed without needing a separate web server.

Chapter 5

Testing

5.1 Test data

5.1.1 Existing collections

The actual test data used for functionality testing is largely based on existing documents, but have been manually processed into CTXT usage context files. Various sources have been used, but mostly, image sets from the Wikimedia foundation have been used. There are several advantages to using these: First, there is the issue of copyright. As Wikimedia tries to include only those images which do not have such restrictions, one can be relatively certain that the images used do not have any restrictions on them, and can be used freely. Second, individual images are part of homogenous categories with brief descriptions. These categories have the properties of collections: a set of images with a usage context, which in this case is the title and brief description of the category. Third, some of the images are part of several different categories, while others are not. This is exactly the type of data this image search system is designed to take advantage of. Since the images in different categories are identical data-wise (they are the same file), they will be recognized as such by this system.

Images in these collections are stored uniformly stored and relatively easy to access, although they cannot easily be retrieved automatically, as Wikimedia prohibits automatic downloading of images by blocking non-browser HTTP requests. This makes gathering test data somewhat time-consuming, as it has to be done manually, but this is not a big problem when using only a relatively limited number of images. Another issue is that many of these collections are not annotated, so the information is not always taken from the category's textual description unedited, but has sometimes been created manually based on category information and image content. In other words, the usage context descriptions generated from Wikimedia collections are not really fair representations of what usage context descriptions based on the available collection information would actually look like. Instead, they are intended mainly for functionality testing and demonstration purposes.

5.1.2 Randomly generated data

To test some performance aspects of the system, some random and relatively diverse data was needed. A simple random word generator was used for this purpose. This generator takes an arbitrary text file, then generates a list of all the unique words in that text file that is both recognized by WordNet in stemmed form and is longer than three characters. These words were then used to generate word-pairs for the testing of WordNet as well as random context descriptions to insert into the database in order to test how well the system could handle larger amounts of data.

5.2 Test environment

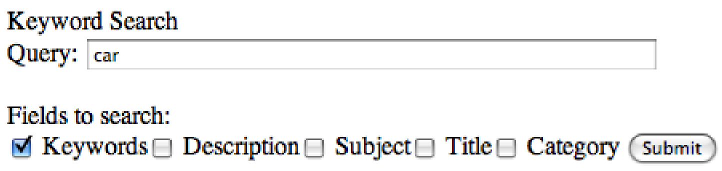
When testing, the front-end of the system was run on a Macintosh computer with a 2-33 GHz Intel Core 2 Duo processor with 2 GB of RAM, running OS X version 10.5.5. The database ran on a separate computer, equipped with an Intel Pentium 3.00 GHz processor and 1 GB of RAM, running Windows XP. The two computers were connected over a local area network, with a latency between the two of around 0.2 ms. All tests were run from the first computer; the database computer was never interacted with directly, only via network connections.

5.3 Functionality

The most basic test of the system is to demonstrate that it works, that it can perform the basic tasks described in the previous sections. This will be shown here through some simple example queries. While by no means representing a thorough functionality test, these examples should serve to give a tangible impression of how the system actually works.

5.3.1 Query 1: Single keyword

The first example is a simple query, using just the search word 'car'. The search is restricted to just the keyword field, as shown by the checkboxes in the screenshot of the web interface in figure 5.1. Entering 'car' in the query box, then clicking the 'submit' button starts the search.


Keyword Search
Query: car

Fields to search:
☑ Keywords ☐ Description ☐ Subject ☐ Title ☐ Category

Figure 5.1: *Query interface.*

The results of this query are shown in figure 5.2. All images in the result belong to the same collection, which has a usage context description titled 'car racing'. It is the only collection in the database having the 'car' keyword, and so no other collections were retrieved by the text search. However, the image at the top also belongs to another collection, containing images of Volkswagens. This collection has a usage context description with the keyword 'automobile'. Since 'automobile' is a synonym of 'car', a semantic link between these two collections was created when the collections were added. The word 'car' is part of the semantic link data, and so the link is judged to be relevant to the query. Therefore, the semantic link strength is added to the 'car racing' collection's relevance score for that image. This shows how semantic links can promote images with content related to the query.

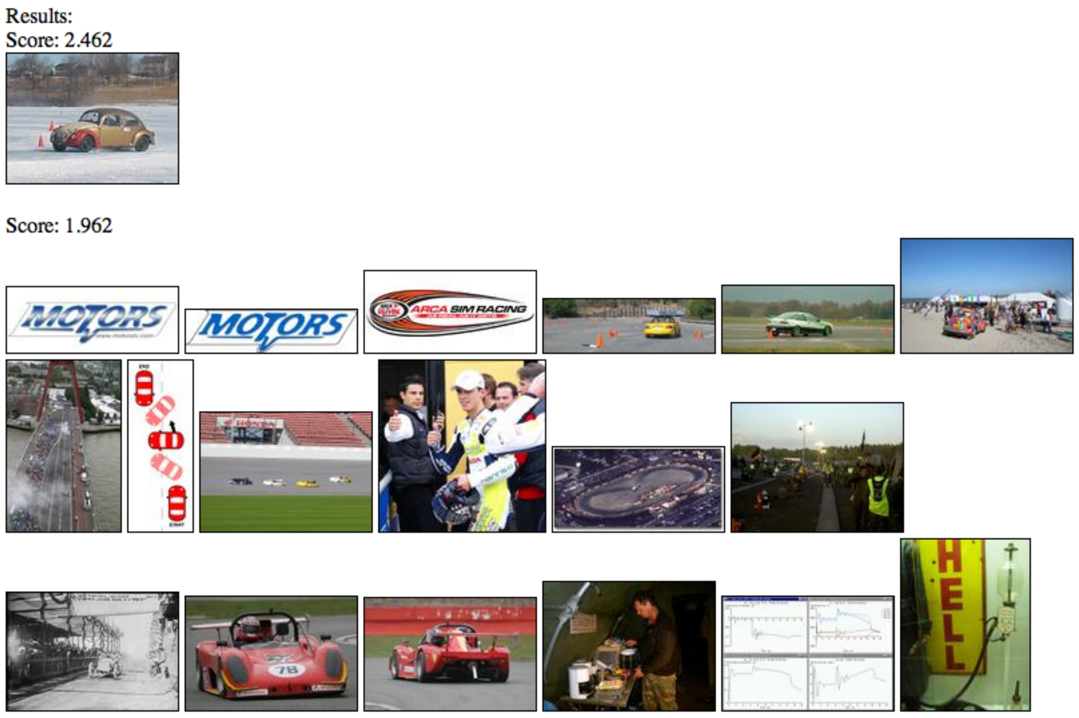


Figure 5.2: *Results for query 'car'.*

5.3.2 Query 2: Two keywords

The second example is a search with the query string 'cicero bust'. In the set of test data collections, there is one usage context description with the keyword 'cicero', a collection containing various depictions of Marcus Tullius Cicero, and one with the keyword 'bust', containing various ancient Roman busts. There is one image that is present in both collections (the other busts in the Cicero collection are modern reproductions). The search returns the results shown in figure 5.3.

Here, both collections are rated as only partially relevant, having a relevance score half of the maximum of 2 before size modifications, as they each only contain one of two search terms. Most of the images in the collections end up with this score. However, the image present in both has the relevance score of both collections, and is therefore displayed first. The image is both a bust and related to Cicero, and so represents an image more relevant to the query than many of the others. This example shows how collection information from two collections can be applied to the same image, allowing a potentially more relevant image to be displayed first.

5.4 WordNet performance

While WordNet is a powerful tool, allowing one to explore the relationships between any two words, doing so is a time-consuming process, since just looking at the distance between all the meanings and synonyms of two words can take a significant fraction of a second. When this operation is repeated several times, as would be necessary in this system, testing has shown that a large delay would be created when any significant amount of data is involved. The test data shown in figure 5.4 illustrates why use of WordNet in a real-time setting is not practical.

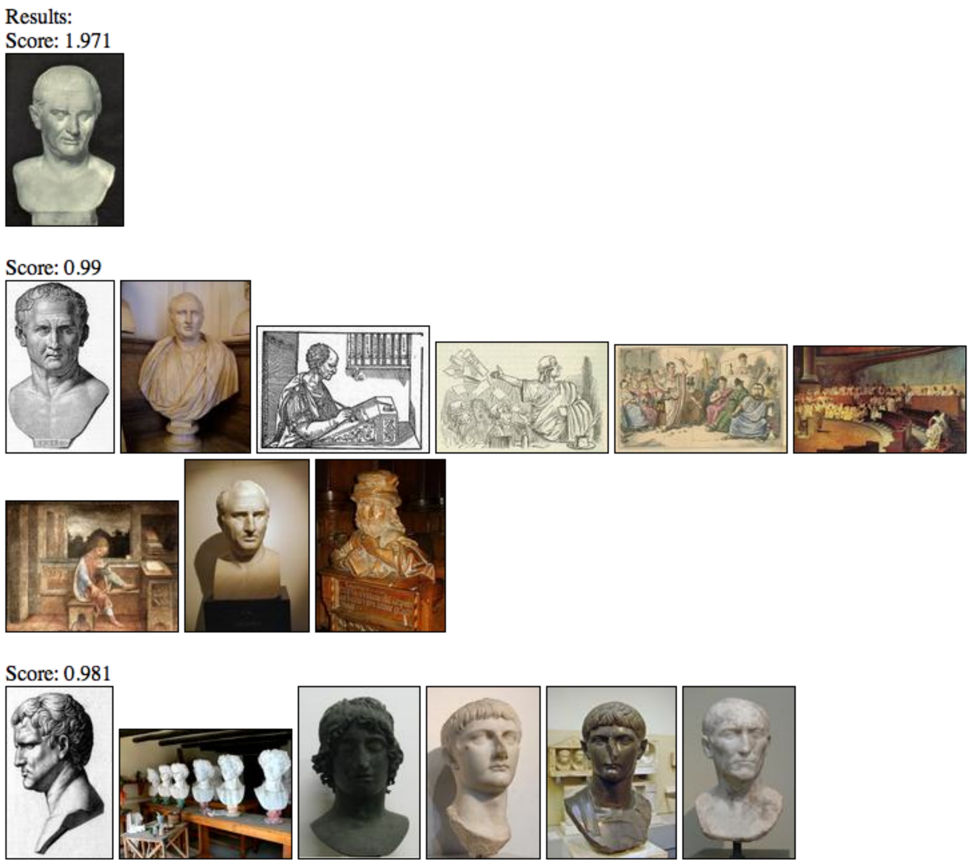


Figure 5.3: *Partial results for query 'cicero bust'.*

Figure 5.4 shows the time it takes to calculate the distances between all synsets of a set of 500 randomly selected word-pairs. A few outlying word-pairs with several hundreds of connections, taking more than 1.5 seconds, have been left out. The average time used is almost exactly 0.4 seconds per word-pair. The number of words needing to be examined in this way between two usage context descriptions would depend on which fields are to be used, but even if just the keyword field is checked for semantic links, one would almost always end up with several word-pairs. If this operation is in turn to be used on several different pairs of collections, it will quickly become too time-consuming. Also, this is an extremely processor-intensive activity, consuming all available processor time when running. It would therefore scale extremely poorly if multiple queries were to use it at the same time.

From the above, it is clear that doing distance calculations in WordNet cannot be done in real-time with the currently available algorithms. However, it must be noted that the current algorithm used is not optimal. The currently available similarity-calculating algorithms available for WordNet in the NLTK will all keep running until the similarity is calculated. However, this system is not really interested in the precise similarity value when it drops below a given value, a value that will be reached quite quickly if the two words are not relatively close. Since calculating distances in WordNet is basically examining a set of interrelated nodes, it should be possible to make an algorithm that stops if the second node is not found within a given distance of the first. Such a method could potentially be quite a lot faster. Implementing it here is beyond the scope of this project, however, as the method and data structure used in the basic algorithm in the NLTK does not lend itself well to this approach.

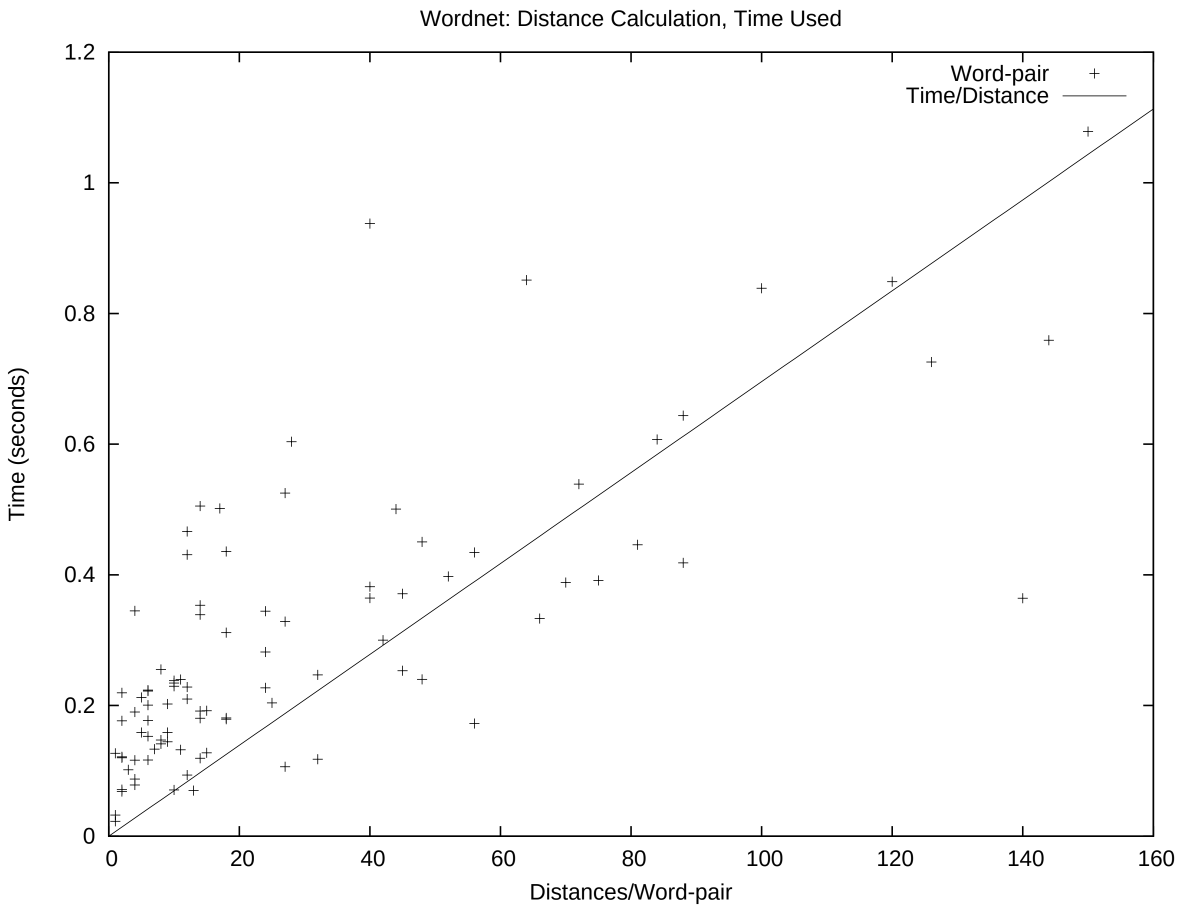


Figure 5.4: *Time used for calculating all distances between the synsets of word pairs.*

5.5 Performance - response time

Measuring the system's response time is interesting here primarily to see how well the system will scale. The absolute values are not really interesting, as they will depend on the hardware of a deployed system, and the performance shown here could likely be improved a great deal through optimizing code and database improvements. Seeing how the system responds to increasing amounts of usage context information in the database can give a good indication of its scalability, however.

5.5.1 Without pre-built text search index

Currently, the text search index used for text searching is created as part of the search operation itself. While not a problem when using only small amounts of data, figure 5.5 shows how the delays associated with the search operations increase as the amount of usage context descriptions stored in the system grows. These tests are run using only the keyword field, with a single search word returning a single result. The times measured are for the actual database queries only, not the rest of the processing in the system. Similar data was also gathered for searching in all fields simultaneously. The delays were then even greater; the last test, for 1 000 000 entries, would cause the web interface to time out, preventing the test from being run to completion.

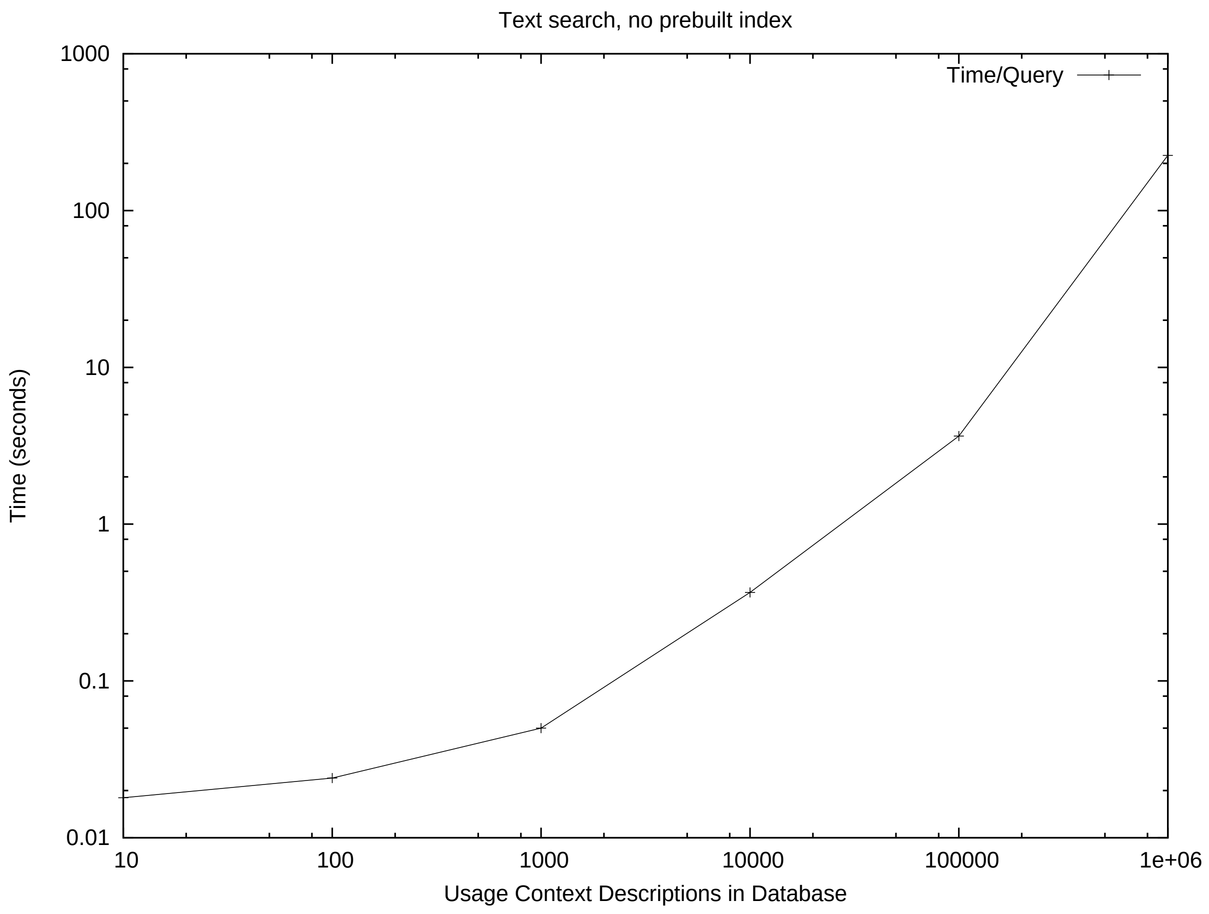


Figure 5.5: *Time used for executing search with no pre-made text search index, logarithmic scale.*

The results from these tests clearly indicate that the system would need to be changed to use pre-made indexes if significant amounts of data were to be added. This process is briefly addressed in the next chapter.

5.5.2 With pre-built index

Manually creating a text search index prior to the search operation can be done through a standard SQL statement. Doing so, then running the same searches as above shows that the big delays in the previous test were overwhelmingly due to the time it took to create the text search indexes. Even when searching a million entries, the time used is below one second. While the time used will likely go up when the load on the database increases, the standard PostgreSQL text search solution used here is still able to cope quite well even with very large amounts of data.

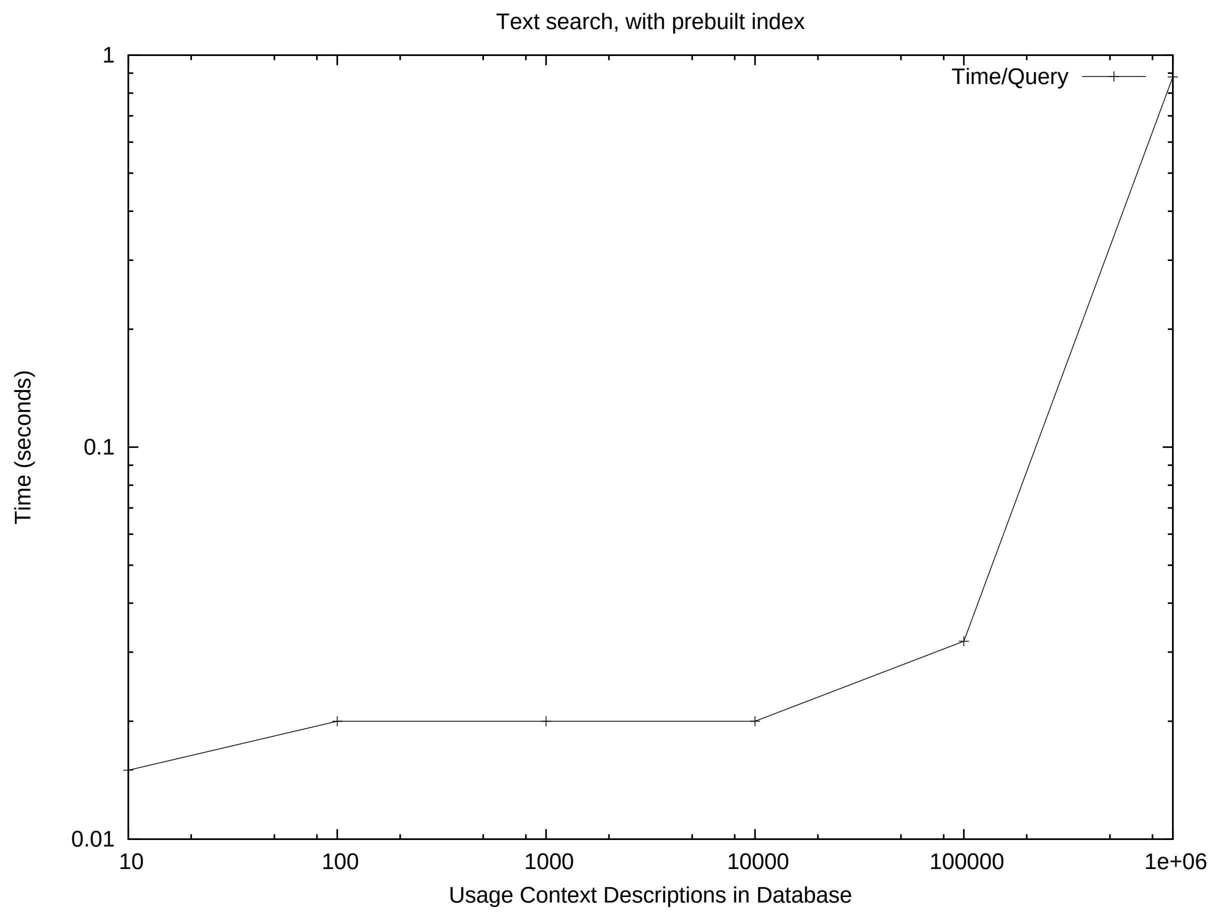


Figure 5.6: *Time used for executing search with pre-made text search index, logarithmic scale.*

5.5.3 Large numbers of results

The two tests above do not measure the performance of the non-database parts of the system, where the information retrieved from the database is processed. For queries with just a few results, the time used is trivial, typically less than 0.001 seconds, even with the relatively inefficient code in place now. However, when the number of results increases, so does the time used to process the results for ranking. No systematic study has been done of this, but some tests have been made. The time used increases steadily with an increasing number of results; 1000 collections in the result will take a bit under 3 seconds to process, while 2000 will take between 5.5 and 6 seconds.

Chapter 6

Conclusion & Future Work

6.1 Conclusion

As has been shown, the system is able to perform image searches where usage context information from different collections are used to rate individual images. Much more testing is likely required to verify that this concept can work using real-world data. However, it has been demonstrated that this approach can serve as the core of an image search system, and that such a system can deliver useful results. It has also been demonstrated that the system should be able to handle greater amounts of data after some modifications. The main goal of this project, to implement a system using image collection which uses the collections' usage context information to search for images, can therefore be said to have been achieved.

As such, while there are many flaws and areas needing improvement in the current implementation of this system, the basic design seems to be sound. However, as mentioned, there is a need for more testing of the system. In particular, no retrieval performance tests have been performed, and so it is not clear how this system stacks up against other information retrieval systems able to search for images. The lack of such tests is quite possibly the greatest flaw of this project, as it is hard to know if a system such as this can really offer an improvement in image search performance.

6.2 Reflections

The lack of focus on retrieval testing seems to be the biggest mistake of this project. In the end, scientific work is only as good as the results it produces. The lack of a good basis for comparisons with other image search systems makes it hard to draw any firm conclusions about the actual value of this system. Even if a well-defined measure of retrieval performance comparable with standard information retrieval algorithms would have proved too difficult, some more basic functional comparison with another system would still have been valuable. There are, of course, reasons for the lack of retrieval testing; some of these challenges are discussed in the Future Work section. However, it is possible that with more work a good comparison with other systems could have been made.

There are also a lot of other testing which should have been done, but could not be performed due to the primitive state of the system. Functional testing with larger data sets is one example; an important concept in this project is that for the system to work well, lots of data should be available. Images that are part of many contexts, and whose semantic content is therefore well-defined, should in theory 'float to the top' of the rankings as a result of this when the images are found to be relevant. It would have been interesting to see if this would indeed have been the case if real-world data had been used. However, the lack of any way to automatically generate usage context descriptions from actual documents meant that preparing truly large amounts of 'real' collections for testing would be far too time-consuming.

Another problem with this project is that formal information retrieval theory has had relatively little influence on the system's design. Part of the reason for this was that the system was a relatively untried concept, and was designed from the ground up based mostly on experimentation rather than a firm plan from the start. The ranking procedures were therefore tacked on more to serve as a proof of concept than because they were the algorithms best suited for the job. While replacing these algorithms is not really difficult, this was in hindsight a waste of a good opportunity for including more theoretical background, which in this project is sorely lacking.

6.3 Future Work

6.3.1 Functionality

There are a number of improvements that could be made to the system at a relatively low level, features that would not impact the basic design of the system. Several of these features were originally planned, but were not implemented due to other features being prioritized. These features would mainly improve the performance, user-friendliness and flexibility of the system. As they are relatively small changes, they are not very interesting to discuss extensively, and will therefore be discussed relatively briefly here. General improvements in the program's reliability, like improved exception handling, would also be desirable, but is not really interesting enough to merit discussion.

Interface improvements

All interfaces currently implemented are quite rudimentary, and not really suitable for non-experimental use. Having a better interface with more options would not only make the system more attractive and easy to use, however; it might also let one find other interesting ways of using this system, or a more advanced one like it. More search options could lead to new and interesting ways of combining collection information when searching. Having a better interface for displaying search results, with e.g. different ranking options, collection browsing functionality and the option of doing refined searches within the results could allow one to explore different ways of exploring image collections. A better storage interface could let one add (and remove) large sets of different collections more easily, allowing one to see the results of searching in different types of collections.

Such interface improvements can require some time to implement, and need a relatively well-defined system in which they can be implemented. Neither of these were available for this project. However, if this system was to be improved, or another, similar system was to be constructed, a better interface would let one have a better idea of what such a system could ultimately do as well as what its limitations would be.

Synonyms

WordNet can be used to retrieve all the synonyms for the different senses of a given word. This can be used to execute searches for the synonyms of any of the words in the query. Since WordNet is more than a simple dictionary, there are several ways one might potentially expand a query using the ontology. All methods discussed here will focus solely on expanding individual search terms; no combining of different search terms will be used. One could, for instance, imagine an algorithm comparing the individual search terms in a query, looking at the distance between various meanings for each term and in this way trying to discover which word sense is being used, but such algorithms are quite complicated and would likely require substantial work.

The first and simplest way to use the ontology is to just find all synonyms for each search term, then run searches for them as well. If such searches return hits not already found by the original query they can be added to the results with a lower weight than the original search term used. This weighting would probably best be based on the number of synonyms and word senses the original search term has; results for the synonyms of a word with many potential meanings should count for less than the results gotten by searching for the synonyms of a word with few alternate meanings, since one in the latter case is generally more likely to get results pertinent to the meaning of the original query.

Second, one can try to search for words that are above or below the search term in the hierarchy. This should be done with some care, since words with many relations in either direction could potentially generate a large amount of irrelevant queries. When using both this and the previously discussed method one also has to make sure that results for synonyms that might be unrelated to the original query do not overwhelm the results for the search terms that were originally entered. One could easily imagine a scenario where some images, though highly relevant to the query, would be ranked far below irrelevant results because synonyms unrelated to the query's intent returned more hits than the original query. One must therefore ensure that these types of query expansion serve as an augmentation rather than a replacement of the original query. To allow this, one must separate such secondary query results from any primary results, so that the presentation module can later assign the secondary results a lower importance than the primary ones. It is also obvious that there is no reason to include a collection in this secondary set if it is already present in the primary results.

Performance improvements

For the reasons outlined in the previous chapter, where database performance was shown, permanent text search indexes are a necessity when the amount of usage context descriptions grows beyond a certain point. While this might seem to be an easy solution, there are a couple of things one has to bear in mind. First, while creating the index is easy and relatively quick outside a real time context, creating it is not enough. The indexes will also have to be updated whenever new data is added to the system. This is the primary reason permanent indexes are not currently used; creating a robust system for maintaining the indexes was not a priority as long as only a few test collections were present. Aside from the work of implementing such index maintenance routines, one also has to check the impact on performance frequent index rebuilds would have. However, as long as these challenges can be met, it seems the basic design of the system is sound for large amounts of data.

Adding collection information to exported images

One idea presented in [10] is to allow for the embedding of collection information in images, for instance through inserting collection CTXT data into image files using the EXIF format. This information can then be part of the image even when it is not directly a part of the system. A further possibility is then to allow images to accrue further CTXT information when used elsewhere, then allow this information to be incorporated into the system at a later time. While creating a system for synchronizing such information is likely to be a significant project in itself, storing CTXT information in EXIF format as part of images shown to the user should be relatively easy to implement as a first step.

Better ranking systems

As has been mentioned, the ranking algorithms used in this system are rudimentary at best. Both the semantic link generator and the usage context description relevance calculator could stand a great deal of improvement. For semantic links, there are a number of algorithms available in the NLTK alone that might be better for establishing similarity. The only way to really determine which algorithm us best would be to test it out on larger data sets, and see which one produces the most reasonable results. Experimentation using larger amounts of data would also likely be needed to determine which ranking algorithm works best in practice, but there can be little doubt that a better algorithm than the makeshift solution currently used is available in the literature.

6.3.2 Retrieval testing

As has been mentioned already, one of the greatest shortcomings of this project is that the actual retrieval performance of this system has not been tested in any formal manner. Part of the reason for this is that performing such tests is a complicated matter, as this system operates on different types of data sources and searches in a different manner than other search systems. While there may well be other systems out there using image collections in the manner of this system, none have so far been found. Almost all existing search systems focus on individual images, rather than collections of them. Setting up a fair comparison test is therefore difficult, as the systems to be compared cannot use identical data sets.

The way this system uses collections also makes it hard to set up meaningful tests based on test data sets created by human experts. The approach used in this project is not designed for accurate retrieval of specific pieces of data, but relies on combining several different pieces of information to get more accurate information for some of the images in a given result set, selecting those with the best information available. This makes it hard to create a single searchable data set that creates non-trivial test situations, while at the same time having an unambiguous optimal search result defined by a human expert. So far, no good way has been found to allow the creation of meaningful human-defined result set that can be compared to the query results from a search in this system.

One possible test scenario is to compare the search performance of this system with another image search system. If the collections could be converted to a format readable by these other systems, a basic functional comparison could be made. One possibility considered, but not implemented due to lack of time and uncertainty as to the value of the results, was to generate web pages based on the collections, have them indexed by Google Image Search, then compare the results with the system presented here. While there does not seem to be a clear specification of exactly how Google Image Search operates, such a test should give some impression of the performance of this system relative to an existing image search solution. One could also try the same with any number of other web image search engines.

6.3.3 Better handling of XML/CTXT data

The problem with storing XML data in a relational database has already been discussed in the design section. As mentioned there, the extensible nature of XML documents means that it can be hard to store them in a standard relational database. If the data from the XML document is simply stored as a standard table with defined fields, there will be nowhere to store values in the XML document that do not have their own fields in the database. While there are a number of approaches possible to handle this problem, as discussed earlier, there is no optimal solution allowing for both full flexibility and rapid text search in the extended fields.

It is questionable whether fields not part of the CTXT standard will have any real value when doing large-scale searches. One of the basic ideas of this system is that a larger quantity of less specific information, gathered from different collections, can make up for a lack of specific data about the individual images that make up the data of interest in the system. It is doubtful that any user-defined fields will have broad enough acceptance to be of use in this setting; it will simply not be present in enough collections to matter for the vast majority of searches. In essence, a user making a query will have to know about the field's existence on his own, since listing all user-defined fields in an interface could quickly become impractical.

6.3.4 Recognizing identical images

As of now, the system only treats images that are identical in every way as the same. This is a neat and simple solution, but not really adequate for real-world use. It is quite common for images to be resized, have their formats changed, and have non-visual information such as attached metadata changed. The images will still be largely identical; in particular, their semantics will not have changed, as they will still be essentially visually identical. Having a way to find images that are semantically identical would be useful for a system such as this. Since the approach taken in this project relies on having several usage contexts descriptions for the same image, increasing the number of images that are identical should lead to improved search performance.

While there may not be any single, integrated, well-developed way of performing this task, this is the sort of task that CBIR can tackle rather well[7], as it relies on comparing features between images. The question is how similar images should be before they are deemed identical. Since format changes and resizing will introduce some visual changes in images, it will not be possible to determine exactly which images are identical and which ones are just very similar; there is no way to know if the minute differences between two images were there when they were created or if they have been introduced later as the result of format changes to a common origin image.

6.3.5 Decentralization

Depending on the amount of information added to the system, at some point, storing all images centrally might become too resource-intensive. One would then have to implement ways for users to access the collection sources directly, instead of hosting the full-size images in a central database and serving them to the user. Some of the problems with a decentralized approach have already been discussed in the Design chapter, and going for a fully decentralized solution might not be feasible, as some images may have to be stored centrally in order to allow users access to them. However, for some collection sources, such as web pages, not hosting the full-size images locally should not prove much of a problem.

6.3.6 Collection quality rating

Another issue alluded to in the design chapter is the problem of knowing how well a usage context description actually describes the images in its collection. The only real way of addressing this is to rate collections or usage context descriptions in some way, either by removing unreliable collections from the database or assigning them an intrinsic weight. Two possible solutions, manual rating of collections and manual rating of collection sources, has already been discussed and largely dismissed in section 3.2.5.

A third possibility is to have ratings be generated by whatever system might eventually generate usage context descriptions. These ratings could indicate how likely the usage context description is to be correct based on e.g. the variety of subjects covered in the usage context and the total amount of information available. Another, more limited method could be to use CBIR to gauge how homogenous images in a collection are, based on size, color, shapes, etc. If a collection contains a lot of images recognized as similar in this way, it is likely that the images are all related, and the likelihood of irrelevant images are lessened.

6.3.7 Collection changes over time

Another problem not addressed in the system is the question of changes in collections. All collections are assumed to be static once entered into the system, but in practice this will of course not be the case. Many of the documents the collections are based on will be changed or re-released with updates, and these changes should be added to and reflected in the system. However, doing this is not a trivial task. Currently, there is no support for time and date information in the CTXT specification. While this could easily be added, it is not as obvious how it would be used usefully. Several different solutions could be proposed, but in the end, the important decision that would have to be made is whether old data should be retained or not. It would be a relatively simple task to simply update a context with new information, adding any new images and removing links to old ones. While this would mean losing older context information, it is questionable whether the extra data and the additional work needed to incorporate this information usefully would be worth the effort. For now, the assumption is that collections will simply be removed and then updated to the newest version if this problem should come up.

Bibliography

[1] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 3–28. Springer, Berlin, 2004.

[2] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[3] R. Datta, Weina Ge, Jia Li, and J.Z. Wang. Toward bridging the annotation-retrieval gap in image search. *Multimedia, IEEE*, 14(3):24–35, July-Sept. 2007.

[4] Ritendra Datta, Dhiraj Joshi, Jia Li, and James Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv.*, 40(2):1–60, 2008.

[5] Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. *The what, who, where, when, why and how of context-awareness, 2000 Conference on Human Factors in Computing Systems*, 2000.

[6] Christiane Fellbaum, editor. *WordNet : an electronic lexical database.* MIT Press, Cambridge, Mass, 1998.

[7] J.J. Foo, R. Sinha, and J. Zobel. Discovery of Image Versions in Large Collections. *LECTURE NOTES IN COMPUTER SCIENCE*, 4352:433, 2007.

[8] Nicholas Galbreath. *Cryptography for Internet and Database Applications: Developing Secret and Public Key Techniques with Java.* John Wiley & Sons, Inc., New York, NY, USA, 2002.

[9] A. Hotho, A. Nürnberger, and G. Paaß. A brief survey of text mining. *GLDV-J. for Computational Linguistics and Language Technology*, 20(1):19–62, 2005.

[10] Randi Karlsen and Joan Nordbotten. Combining image context information. In *Norsk informatikkonferanse : NIK 2008 : Institutt for informasjons- og kommunikasjonsteknologi, Universitetet i Agder, 17.-19. november 2008*, Trondheim, 2008. Tapir akademisk forlag.

[11] Raymond J. Mooney and Razvan Bunescu. Mining knowledge from text using information extraction. *SIGKDD Explor. Newsl.*, 7(1):3–10, 2005.

[12] Arnold W. M. Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-based image retrieval at the end of the early years. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(12):1349–1380, 2000.

[13] Ka-Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. Faceted metadata for image search and browsing. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408, New York, NY, USA, 2003. ACM.

[14] Rong Zhao and William I. Grosky. *Bridging the semantic gap in image retrieval.* Hershey, PA, USA, 2002.